Санкт-Петербургский государственный университет
Экономический факультет
Кафедра информационных систем в экономике

Кауров Андрей Алексеевич

Выпускная квалификационная работа

Модель выбора методологии разработки программного обеспечения

Направление 38.04.05 «Бизнес-информатика»
Основная образовательная программа магистратуры «Информационная бизнес-аналитика»

Научный руководитель:
Кривцов Александр Николаевич,
Кандидат физико-математических наук, доцент

Рецензент:
Киселев Дмитрий Игоревич,
Генеральный директор
ООО «Клевер Солюшнс»

Санкт-Петербург
2018 г.

Оглавление

Введение

Современное состояние проблемы

Исследования подходов разработки программного обеспечения (ПО) являются важной и актуальной частью исследований в области программной инженерии. Повсеместное внедрение информационных технологий способствует появлению большого внимания к исследованию наиболее эффективных методов разработки ПО. На данный момент существует большое количество научных работ, посвященных изучению особенностей методик разработки ПО и сравнению их между собой. Так, например, исследования Ильясовой Ф.С. «Современные методологии разработки программного обеспечения»[7] и Аникеева Д.А «Обзор методологий разработки программного обеспечения» [1] посвящены детальному изучению популярных методик разработки: каскадной модели, спиральной модели, компонентно-ориентированной модели, RUP (рациональный унифицированный процесс), MSF (Майкрософт фреймворк-решения), XP (экстремальное программирование), OpenUP (итеративно-инкрементальный метод разработки ПО), FDD (особенности управляемого процесса), Scrum (фреймворк гибкой разработки ПО).

Особенно остро стоит проблема выбора той или иной методики разработки ПО в зависимости от особенностей решаемых задач и требований организаций, в следствие чего огромное количество трудов посвящено этой тематике. В работе Бурбело С.М. [3] «Выбор гибких методов разработки программного обеспечения» и в исследованиях других авторов на эту же тему, например, таких, как Карпов Д.В. [9], Тюнина А.И. [22], Борцов М.Ю.[2], Лим С.А. [10], Сербская О.В. [19], Восканян Л.С. [4] акцентируется внимание на изучение особенностей использования гибких методик разработки программного обеспечения и обоснования их преимуществ. Эти исследования теоретически доказывают превосходство гибких методов, но не имеют за собой достаточно убедительного практического обоснования. Проблематика анализа и выбора наиболее подходящих методов рассматривается в таких работах, как «Анализ процесса выбора технологии проектирования, методологии и среды разрабатывания программного обеспечения» Говорущенко Т.О [5], «Анализ современных методологий разработки сложных программных проектов» Свиридова А. С. [20], «Сравнительный анализ методологий разработки программного обеспечения» Геркушенко Г. Г. [21]. В этих трудах предпринимается попытка создания некоторой модели выбора того или иного метода в зависимости от имеющихся условий. Кроме этого стоит отметить работы Михайлова А. А. [13], а также статью «Современные технологии производства и перспективные модели

разработки программного обеспечения» Плотникова А.Н. [17] отличающиеся детальным изучением отдельно взятых методик.

Часто исследователи сужают работы до какой-то конкретной сферы или задачи, так, например, в статье «Проблема выбора методологии разработки информационной системы вуза» Петровой А.Н.[16] рассматривается ситуация отбора методики для внедрения ИС в высшее учебное заведение Многие работы посвящены использованию тех или иных методик в бизнесе: так, исследование под названием «Применение методологии Scrum в передовых компаниях» Восканян Л.С.[4] изучает существующие практики применения методологии Scrum, изучает ее плюсы и минусы. Аналогичное исследование проводится в работе «Проблемы адаптации scrum-инструментов в российской практике управления ит-проектами» Евсеева Л.В.[6].

Многие работы посвящены усовершенствованию и дополнению уже имеющихся методик с целью повышения их эффективности на практике. Примером такой работы служит статья «Применение имитационного моделирования в управлении проектами по разработке программного обеспечения с гибкими методологиями» Сачек Е.А.[18], аналогичная работа, связанная с повышением качества разработки на основе анализа проектных рисков, опубликована Овчинниковым С.А.[14]. В другой работе, «Инновационные методологии перепроектирования «унаследованных» программных систем» Матвиенко Д.М.[11], приводится описание опыта команды разработчиков, решающей задачу внедрения механизмов тест-ориентированной разработки (Test-Driven Development) в существующий проект, первоначально не адаптированный к использованию ТОО-инструментария.

Таким образом, можно резюмировать, что к сфере разработки программного обеспечения проявляется большой научный интерес, множество специалистов пишут исследовательские работы в данном направлении, но остается большое количество открытых вопросов в отношении методов разработки ПО, что обуславливает пространство для дальнейшей работы.

Сегодня можно констатировать наличие огромного количества различных методов, описываемых различными принципами и требованиями. Такое количество различных способов разработки является результатом исследования специалистов с целью создать максимально эффективный метод под ту или иную задачу. Из-за этого встречается большое количество различных методов «симбиозов», объединяющих как плюсы, так и минусы нескольких методик в одной.

Это является большой проблемой для предприятий, стремящихся иметь статус высокотехнологичных и инновационных. В деятельности таких предприятий большую

роль играют новые качественные технологии и научные исследования. Задачи, решаемые высокотехнологичным предприятиями, могут быть совершенно разными, это касается и процесса разработки программного обеспечения: здесь задача может стоять как в внедрении какого-то специализированного приложения для работы конкретного сотрудника, так и в развертывании сложной корпоративной системы, имеющей большое количество различных интегрированных приложений. Одной универсальной методики разработки ПО под все эти случаи не существует, выбор же из огромного множества какой-то конкретной, отвечающей представленным требованиям, очень серьезная и затратная задача.

Это **обуславливает актуальность** проводимого исследования, направленного на изучение и сравнение наиболее часто используемых методов проектирования ПО и создание рекомендационной модели выбора методологий в зависимости от требований для различных задач.

Цель работы: выявить модель выбора методологии разработки программного обеспечения в зависимости от исходных условий и требований к проектной деятельности.

Гипотеза исследования: если существует ряд отличающихся друг от друга методологий разработки программного обеспечения, то должна существовать возможность сформулировать алгоритм выбора какой-то определенной из них. Например, модель, основанная на экспертных оценках и моделях минимизации риска.

Задачи исследования:

1. Проанализировать существующие методики разработки IT-проектов;
2. Определить критерии оценки методик разработки программного обеспечения при постановке той или иной задачи;
3. Разработать модель выбора методики в зависимости от установленных ограничений;
4. Проанализировать и получить оценку состоятельности модели для рассматриваемого ряда задач по установленным критериям;

Объект исследования: система методологий и подходов по разработке программного обеспечения**.**

Предмет исследования: методологии разработки IT-проектов в деятельности высокотехнологичных предприятий.

Научная новизна. Научная новизна в исследовании заключается в разработке модели выбора и создании методических рекомендаций по совершенствованию анализа подходов к разработке программного обеспечения в высокотехнологичных предприятиях.

Теоретическая значимость. Теоретическая значимость работы заключается в рассмотрение фундаментальных и методологических основ разработки программного обеспечения.

Практическая значимость. Практическая значимость исследования заключается в возможности использования разработанной модели и рекомендаций для совершенствования проектной деятельности в высокотехнологичных компаниях с широким набором IT-проектов, например, в структурах компании «Газпром нефть».

Структура выпускной квалификационной работы. Магистерская работа состоит из ведения, трех глав, заключения, списка использованной литературы.

Во введении обоснована актуальность темы, определены цель, задачи, объект, предмет, информационная база, теоретическая и методологическая основа, раскрыта научная новизна, теоретическая и практическая значимость работы.

В первой главе «Анализ современного состояния вопроса выбора методологий разработки программного обеспечения» определяется понятийный аппарат предметной области, после чего проводится подробный обзор и анализ некоторых популярных методологий разработки программного обеспечения, их преимуществ и критериев использования

Во второй главе «Существующий инструментарий выбора методологии разработки ПО» исследуются предпосылки использования методологий разработки, рассматривается концепция Кеневин, позволяющая определить критерии оценки проектов по разработке программного обеспечения и классифицировать их. Предлагается и теоретически описывается модель, позволяющая анализировать сопоставимость той или иной методологии с различными проектами

В третьей главе «Практическое применение модели выбора методологии разработки программного обеспечения» приводится результат практического применения разработанной модели в качестве инструмента поддержки принятия решения руководителя проекта в компании Клевер Солюшнс. Проводится анализ результатов применения модели и дана оценка ее состоятельности и применимости в реальной проектной деятельности

В заключении сформулированы основные выводы и предложения, полученные в результате проведенного магистерского исследования.

На защиту выносится сформулированная модель выбора методологии разработки программного обеспечения.

Глава 1 Анализ современного состояния вопроса выбора методологий разработки программного обеспечения

1.1 Основные принципы и понятия разработки программного обеспечения

Методологии разработки программного обеспечения являются одной из основных тем, изучаемых в рамках программной инженерии, которая в свою очередь рассматривается как приоритетное направление проектной деятельности высокотехнологичных предприятий.

Под высокотехнологичным предприятием будем понимать предприятие, которое стремится в рамках своей деятельности осуществлять инновационно-креативную политику с целью более эффективного по сравнению с конкурентами выполнения аналогичных видов деятельности. В своей работе «Высокотехнологичное предприятие как субъект инновационно-креативной деятельности» Мерзлякова А.П [12] определяет высокотехнологическое предприятие следующим образом: «высокотехнологичное предприятие должно быть ориентировано на инновационно-креативное развитие для повышения эффективности своей деятельности на основе создаваемых конкурентных преимуществ». Как правило, эта инновационно-креативная деятельность связанна с сферой информационных технологий, например, с такими областями как программирование, автоматизация, роботизация.

Определим, что такое «программная инженерия» и чем она отличается от программирования.

Понятие «программирование» - имеет много значений. Так обобщено называет деятельность, связанную работой с компьютерными технологиями, в частности с написанием программного кода и созданием программного обеспечения. Кроме того, такое понятие может использоваться в образовательных, досуговых целях, когда процесс разработки программного обеспечения представляет собой подобие творческого процесса по аналогии с искусством. Объективная потребность контролировать этот процесс, прогнозировать и гарантировать выполнение поставленных критериев (например, бюджет, сроки и качество), послужила появлению программной инженерии [35].

Программная инженерия в этом случае – формализованный и контролируемый процесс создания программного продукта, включающий в себя большое количество подпроцессов, не связанных непосредственно с созданием кода, так, например, это:

- определение требований;

- написание проектной документации;
- создание функциональных моделей;
- планирование процесса разработки;
- тестирование продукта;
- конфигурационный менеджмент;
- проектный менеджмент и т. д.

Все эти процессы взаимосвязаны друг с другом и вытекают одно из другого. Так в «Руководстве к своду знаний по программной инженерии» (SWEBOK V3.0 – Software Engineering Body of Knowledge) [48] – одному из главных своду знаний по программной инженерии, определяется 15 направлений, исследуемых в области программной инженерии:

1. software requirements — требования к ПО;
2. software design — проектирование ПО;
3. software construction — конструирование ПО;
4. software testing — тестирование ПО;
5. software maintenance — сопровождение ПО;
6. software configuration management — управление конфигурацией;
7. software engineering management — управление IT проектом;
8. software engineering process — процесс программной инженерии;
9. software engineering models and methods — модели и методы разработки;
10. software engineering professional practice — описание критериев профессионализма и компетентности;
11. software quality — качество ПО;
12. software engineering economics — экономические аспекты разработки ПО;
13. computing foundations — основы вычислительных технологий, применимых в разработке ПО;
14. mathematical foundations — базовые математические концепции и понятия, применимые в разработке ПО;
15. engineering foundations — основы инженерной деятельности.

Таким образом, под программной инженерией будет пониматься целая система процессов, направленных на разработку программного обеспечения.

Необходимость контролировать и управлять этими процессами послужила тому, что наиболее удачные опыты и практики построения процессов разработки ПО стали

распространяться и массово применяться, получили свои названия и стали объектами научных исследований многих специалистов.

Так были созданы и сформированы методологии разработки программного обеспечения, описывающие различные подходы к процессам разработки ПО в зависимости от имеющихся условий.

В учебнике «Программная инженерия: методологические основы» В.В. Липаева определяется суть методологии программной инженерии: «применение систематизированного, научного и предсказуемого процесса проектирования, разработки и сопровождения программных средств» [35].

1.2 Существующие подходы к разработке программного обеспечения

В современной программной инженерии определяют следующие формы, определяющие подход к разработке программного обеспечения:

- Принципы;
- Модели;
- Методологии.

Важно обозначить основные различия этих форм.

Принципы и модели программного обеспечения – набор определяющих верхнеуровневых правил или низкоуровневых практик, положений, на основе которых создано большое количество методологий.

Например, к принципам относится известный Agile (гибкая разработка ПО) – набор определенных ценностей, принципов и практик, обуславливающих «гибкий» подход к разработке программного обеспечения. Это не какая-то конкретная методология, а большой набор эффективных практик, которые могут быть использованы как в совокупности, так и по отдельности.

К наиболее популярным моделям относится:

- Waterfall – каскадная модель;
- Incremental – инкрементальная (итеративная) модель;
- Spiral – спиральная модель.

Методологии программного обеспечения – это частные практики, основывающиеся на той или иной модели или принципе, но содержащие конкретное описание действий в процессе разработки ПО. Зачастую, организации стараются создать и анонсировать свою собственную корпоративную методологию, которая порой будет отличаться только одним

специфичным критерием, поэтому на данный момент создано огромное количество разнообразных методологий.

Среди методологий, основанных на принципах гибкой разработки самыми популярными являются:

- Scrum;
- Kanban.

Рассмотрим перечисленные модели и принципы:

1. **Waterfall – каскадная модель**

Одна из фундаментальных моделей, первый раз предложенная в 1970 году [45].

Подразумевает прохождение последовательных стадий проектной деятельности (Таблица 1). Каждая стадия начинается после завершения предыдущей. Такой подход был принят для того, чтобы можно было фиксировать количество требований, изменений и задач: до изобретения каскадной модели повсеместно использовался подход к разработке «code and fix» («разрабатывай и правь»). Подход подразумевал последовательную работу по разработке продукта и исправлению его недостатков, но так как количество недостатков нигде не фиксировалось, а порой с развитием продукта только увеличивалось, сроки разработки могли затягиваться на годы. Каскадная модель позволяет четко фиксировать все требования и разрабатывать именно то, что требует заказчик.

Таблица 1.

Стадии каскадной модели

Стадия	Описание
1. Определение требований (анализ)	На этапе анализа изучается и определяется задача, которую должна выполнять программа. Результатом выполнения этой фазы является совокупность требований, предъявляемых к ПО.

Продолжение таблицы 1

2. Проектирование	На этом этапе требования, выявленные при анализе, преобразуются в описание принципов решения – документ, в соответствии с которым принимаются конкретные решения при реализации программы. Основным итогом второй фазы является получение проекта, который может включать текст на естественном языке, модель ПО, алгоритмы, таблицы, математические формулы и т. п. Детальное проектирование предполагает выделение компонент ПО, определение их структуры и методов взаимодействия.
3. Реализация (разработка)	По завершении исходного проектирования следует этап реализации, на котором создаются и тестируются программные модули, определенные при проектировании. Главными результатами этого этапа являются модули исходного кода и автономные тесты модулей.
4. Тестирование	На этой стадии происходит полное комплексное тестирование работоспособности разработанного продукта и его соответствие требованиям заказчика.
5. Внедрение	Готовый программный продукт передается заказчику, производятся приемо-сдаточные испытания, осуществляется обучение пользователей и опытная эксплуатация.

Процесс управления проектом на основе каскадной модели прост, за это ее любит большинство руководителей. Модель четко фиксирует все стадии и последовательность процессов, словно инструкция, разработка проходит быстро, стоимость и срок заранее

определены: каскадная модель предполагает изначальное составление четкого плана и графика работ и безоговорочное следование ему в процессе разработки. При таком подходе требования заказчика выявляются и утверждаются еще на этапе планирования проекта.

Но в этом и заключаются основные проблемы модели: такой подход будет приносить хороший результат только в проектах с хорошо проработанными и заранее определенными требованиями и способами их реализации, а это большая редкость. При использовании каскадной модели практически отсутствует возможности сделать шаг назад, цена ошибки очень велика. Как правило, продукты, разработанные при помощи без обоснованного выбора каскадной модели, имеют недочеты, связанные с неполным удовлетворением требований или ожиданий заказчика, который сталкивается с продуктом на последней стадии, когда разработка уже завершена и нет возможности внести какие-либо изменения (Рисунок 1-1).

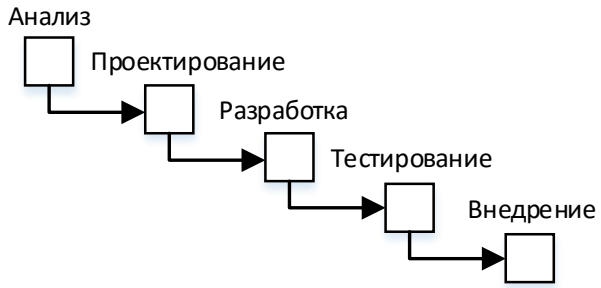

Рисунок 1-1. **Каскадная модель**

Тем не менее, каскадная модель очень популярна, особенно у руководителей, которые имеют поверхностные знания о процессе разработки программного обеспечения. Фиксированная стоимость разработки часто перевешивает минусы модели. Исправление выявленных в процессе разработки недостатков, несоответствий требований и ожиданий возможно, но как правило, подразумевает создания дополнительного соглашения к имеющемуся или подписание нового договора на разработку, который в свою очередь также будет иметь все плюсы и минусы каскадной модели.

2. Iterative and Incremental model – итеративно-инкрементальная модель

Итеративно-инкрементальный подход к разработке ПО берет свое начало с середины 50-х годов прошлого столетия [47]. Но если в те времена понятие «итеративная разработка» сводилась к исправлению уже сделанного, то в контексте современных методов этот термин означает нечто иное: не просто пересмотр проделанной работы, но и эволюционное продвижение вперед.

Итеративно-инкрементальный подход основывается на базовом формальном описании системы, дающем возможность создать первую исполняемую функциональную

модель. Полученная модель проверяется на соответствие описанию системы, а затем расширяется далее, последовательно преобразуясь в новые модели, в которых отражается увеличение требований к системе и уточнение деталей их реализации.

Итеративная модель жизненного цикла подразумевает начало разработки продукта с того функционала, по части которого имеется наиболее полное описание требований. То есть отсутствует необходимость иметь на руках полную подготовленную спецификацию. По мере прояснения требований, в работу включается новая функциональность. Каждая итерация должна быть результативна, а каждая версия продукта – работоспособна.

Итеративный подход предполагает разбиение жизненного цикла проекта на последовательность итераций, каждая из которых напоминает “мини-проект”, включая все этапы жизненного цикла ПО в применении к созданию меньших фрагментов функциональности, по сравнению с проектом, в целом. Цель каждой итерации – получение работающей версии (релиза) ПО, включающей функциональность всех предыдущих и текущей итерации.

Результат финальной итерации содержит всю требуемую функциональность продукта. Таким образом, с завершением каждой итерации, продукт развивается инкрементно.

Таблица 2.

Инкрементальная модель

Стадия	Итерация 1	Итерация 2	…	Итерация N
Определение требований	Шаг 1	Шаг 5	…	Шаг n-4
Проектирование	Шаг 2	Шаг 6	…	Шаг n-3
Реализация	Шаг 3	Шаг 7	…	Шаг n-2
Тестирование	Шаг 4	Шаг 8	…	Шаг n-1
Внедрение	-	-	-	Шаг n

С точки зрения структуры жизненного цикла такую модель называют итеративной. С точки зрения развития продукта – инкрементной.

Рабочий процесс по этой модели принимает эволюционный вид. С каждой новой итерацией продукт принимает все более и более конечный вид.

3. Spiral – спиральная модель

Спиральная модель напоминает инкрементальную, так как она также подразумевает пошаговую разработку, но здесь это делается для акцентирования внимания на рисках.

Кроме того, она напоминает каскадную модель, так как последовательность действий строго определена и фиксирована, большое внимание уделяется формированию требований и ведению документации.

Как правило, модель используется для разработки стратегически важных систем и проектов, для научно-исследовательской деятельности, когда нужно оценивать каждое последующее действие, так как рискованность разработки высока, а последствия неудачи могут быть критичны [25].

Модель определяет 4 действия, представляемыми 4-мя квадрантами спирали.

1. Планирование – определение целей, вариантов и ограничений;
2. Анализ риска – анализ вариантов и распознавание/выбор риска;
3. Разработка – разработка продукта следующего уровня;
4. Оценка – оценка заказчиком текущих результатов конструирования.

Рисунок 1-2 - **«Спиральная модель разработки ПО»**

Основная особенность данной методологии состоит в концентрации сложности на начальных этапах жизненного цикла ПО (анализ, проектирование); при этом сложность и трудоемкость последующих этапов в пределах одного витка спирали относительно невысокие. По этой методологии предлагается способ снижения затрат в целом при разработке ПО за счет предотвращения потенциальных ошибок на этапах анализа и проектирования. Этап определения стратегии присутствует на первом витке спирали либо «склеен» с этапом анализа первого витка спирали.

4. Гибкая разработка программного обеспечения (Agile software development)

Берет свое начало в 2001 году, когда был опубликован «Agile Manifesto» (Манифест гибкой методологии разработки программного обеспечения) [46], в котором были сформированы 4 идеи и 12 принципов «альтернативного» тому времени подхода к разработке ПО:

Идеи Agile:

1. люди и взаимодействие важнее процессов и инструментов;
2. работающий продукт важнее исчерпывающей документации;
3. сотрудничество с заказчиком важнее согласования условий контракта;
4. готовность к изменениям важнее следования первоначальному плану.

Принципы Agile:

1. удовлетворение клиента за счёт ранней и бесперебойной поставки ценного программного обеспечения;
2. приветствие изменений требований даже в конце разработки (это может повысить конкурентоспособность полученного продукта);
3. частая поставка рабочего программного обеспечения (каждый месяц или неделю, или ещё чаще);
4. тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта;
5. проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;
6. рекомендуемый метод передачи информации — личный разговор (лицом к лицу);
7. работающее программное обеспечение — лучший измеритель прогресса;
8. спонсоры, разработчики и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;
9. постоянное внимание улучшению технического мастерства и удобному дизайну;
10. простота — искусство не делать лишней работы;
11. лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды;

12. постоянная адаптация к изменяющимся обстоятельствам. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.[1]

Кроме идей и принципов, agile включает в себя большое количество различных гибких «практик». Практикой, например, является использование «спринтов» или «канбан-доски», то есть это частный и специфический пример использования принципов agile. Набор определенных практик, результативно использующихся вместе, объединяют в «фреймворки» (framework) – комплекс практик и правил, аналог методологий.

Agile-методы делают упор на непосредственное общение сотрудников. Большинство agile-команд расположены в одном офисе, кроме того, там же обычно располагаются «заказчики» (заказчики которые определяют продукт, это могут быть менеджеры продукта, бизнес-аналитики или клиенты). Считается хорошей практикой, когда на время проекта или проектная группа со стороны исполнителя, или группа консультантов со стороны заказчика переезжают в офис для совместной работы.

Далее представлены самые известные и наиболее часто используемые фреймворки гибкой разработки – скрам и канбан.

5. Scrum - скрам

Скрам – это фреймворк, который начал широко распространяться в начале 90-х годов на примере разработки сложных комплексных продуктов.

Скрам подразумевает под собой набор эффективных действующих управленческих и технических практик по разработке ПО, кардинально меняющий представление о процессе разработки. Многие руководители «боятся» скрама, так как понимают, что их организация не готова применять в своей работе такое количество нетрадиционных инструментов и возлагать на сотрудников ответственность самостоятельного выбора действий. Многие ассоциируют попытку внедрения в организацию скрама с революцией, и это не случайно.

Тем не менее, скрам – одна из самых популярных практик применения принципов гибкой методологии. На Рисунок 1-3 представлены данные из ежегодного исследования State of Agile [44], которое проводится компанией VersionOne уже 11-й год среди организаций по всему миру и охватывает все основные аспекты применения Agile.

---

[1] Agile-манифест разработки программного обеспечения [Электронный ресурс]. URL: http://agilemanifesto.org/principles.html/ (дата обращения: 02.11.2017)

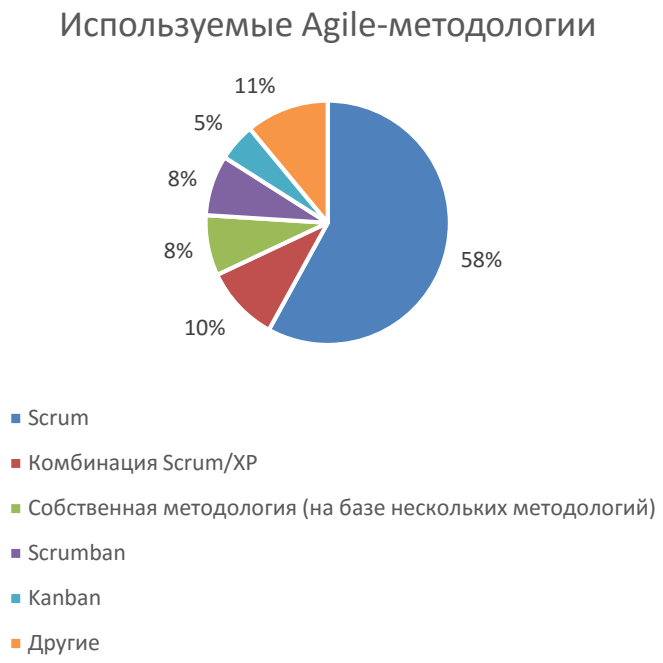

Рисунок 1-3 – **Диаграмма «Используемые Agile-методологии»**

Источник: 11-й ежегодный отчет State of Agile [44]

Основными элементами фреймворка являются:

- артефакты;
- события;
- скрам-команды;
- роли в скрам команде;
- правила.

Каждый из элементов фреймворка, соответствует определенной цели и является обязательным для эффективного использования фреймворка. Зачастую это вызывает сопротивление со стороны заказчика, не готового полностью или строго выполнять какой-либо из элементов скрама. Поэтому основная и особо трудоемкая задача при скрам-трансформации предприятия – довести до каждого сотрудника категоричность выполнения всех правил и инструкций. Если представить скрам как процесс развития, то это революционный процесс изменения всех процессов предприятия, поэтому многие компании опасаются полного внедрения и используют частичные практики.

Артефакты скрама

Артефакты скрама – основные понятия и объекты, используемые в данном фреймворке. Требуется обеспечить понимание каждым участником проекта используемой терминологии и ее содержанием.

1. **Бэклог продукта**

Бэклог продукта – это упорядоченный список всех требований к продукту. Это единственный источник и инструмент внесения требований для любого вида изменений, которые могут быть внесены в продукт. Всю ответственность за содержание, доступность и приоритизацию бэклога несет владелец продукта и только он. Любое, даже самое срочное и критическое изменение может быть внесено только через бэклог продукта.

Особенность бэклога продукта в том, что он никогда не бывает полным. Это отличительная черта позволяет оперативно и без ущерба для процесса разработки реагировать на любые изменения требований.

Как правило, начальный вариант бэклога, когда проект только стратовал, содержит только самые полные и известные требования, что позволяет определить разработчикам сформулированные задачи, которые можно брать в работу, а аналитикам продолжить проработку требований по «непроработанным» задачам. По мере проработки требований, задачи будут добавляться в бэклог, по мере выполнения задач – убираться из него, таким образом, можно наблюдать динамку изменений и работы команды.

Уточнение Бэклога Продукта (PBR – product backlog refinement) – это процесс по уточнению, актуализации, оценке и упорядочиванию элементов в бэклоге продукта. Владелец продукта и команда разработки прорабатывают детали элементов бэклога продукта, тем самым анализируя и подготавливая эти элементы для будущей работы. Как правило, этот процесс планирования и уточнения происходит параллельно текущему спринту. При этом владелец продукта может изменить элементы бэклога продукта в любой момент времени, но это никак не отразится на уже запущенный в спринт бэклог – изменения возможны только после завершения текущего спринта.

2. **Бэклог спринта**

Бэклог спринта – это элементы бэклога продукта, выбранные командой разработки для реализации в текущем спринте.

Бэклог спринта служит неким план-прогнозом команды разработки касательно функциональности, которая станет доступна при поставке следующего инкремента. Для контроля и анализа прогресса разработки используются ежедневные скрам встречи, на которых скрам-команда делится результатами предыдущего дня и возникшими в процессе проблемами.

Ответственность за бэклог спринта лежит исключительно на команде разработки и служит планом работ, который команда обязалась выполнить в течение спринта.

3. Инкремент

Инкремент – это совокупность реализованных за текущий спринт элементов бэклога продукта инкрементов предшествующих спринтов.

В результате завершения спринта команда должна подготовить инкремент к публикации и представлению заказчику.

Спринт

Спринт – механизм функционирования всего скрама, вокруг него построены все процессы. Это циклический процесс реализации бэклога спринта, длительностью от 1 до 4 недель, в течение которого команда разрабатывает функционирующий инкремент продукта.

Продолжительность спринта остается неизменной и четко зафиксированной на протяжении всего проекта. Спринт – это циклический процесс, как только завершается один – начинается другой.

Кроме непосредственно процесса разработки, спринт состоит из:

- планирования спринта – процесс планирования бэклога спринта, участвует вся скрам-команда;
- уточнение бэклога продукта – корректировка бэклога продукта и планирование следующего спринта, процесс идет параллельно текущему спринту;
- ежедневных скрам встреч – синхронизация действий команды, обсуждение выполненных работ и возникших проблем;
- обзора спринта – в обзоре спринта участвует скрам-команда и все заинтересованные лица, на встрече демонстрируют инкремент, собирают отзывы и пересматривают план развития;
- ретроспективы спринта – проводится при завершении спринта, обсуждаются все результаты работы внутри скрам-команды, предлагаются идеи по улучшению рабочего процесса.

Каждый Спринт можно считать небольшим проектом длительностью не более одного месяца.

Скрам-команда

Это самоорганизующаяся, кроссфункциональная и постоянная команда. Команда самостоятельно решает, каким образом и в каком объеме выполнять работу, компетенции членов команды полностью покрывают все предъявляемые к задаче требования, команда постоянна и не распускается при завершении проекта или работ. Это означает, что каждая

эффективная команда – это группа специалистов широкого профиля, слаженно работающих в команде и несущих коллективную ответственность за свою работу.

Скрам-команда состоит из:

- владельца продукта;
- команды разработки;
- скрам-мастера.

Подход работы в скрам-команде предполагает отсутствие внешних зависимостей команды, отсутствие возможности «извне» влиять на решения или действия команды. Такой подход располагает к гибкости, творчеству и продуктивности. Продукт разрабатывается итеративно и инкрементально для чего критична возможность иметь постоянную обратную связь от заказчика.

Владелец продукта

Владелец продукта – представитель команды, в чьи обязанности входит взаимодействие с заказчиком и контроль разработки продукта, приносящего заказчику максимальную ценность. Владелец продукта отражает требования заинтересованных лиц в бэклоге продукта, посредством которого и происходит процесс контроля. Владелец продукта – единственное лицо, имеющее право управлять бэклогом продукта. Управление бэклогом подразумевает под собой описание и приоритизацию его элементов.

Команда разработки

Команда разработки отвечает за создание инкремента продукта наивысшего качества, при этом никто не имеет право ставить требования команде по процессу и объему разработки – команде категорически запрещается следовать чьим-либо указаниям, иначе это уже не скрам.

Все члены команды именуются разработчиками, вне зависимости от типа выполняемых задач или областей компетенции. Как указывалось, ранее, команда разработки имеет следующие характеристики:

1. самоорганизующаяся – члены команды самостоятельно координирует свою деятельность. Такая свобода действий способствует достижению синергии, роста эффективности и производительности, кроме того, команда также самостоятельно решает, как и в каком объеме превращать бэклог продукта в инкремент**.** Никто не может диктовать правила команде разработки, даже скрам-мастер;
2. кроссфункциональная – разработчики могут быть специалистами какой-то конкретной области, но при этом они должны профессионально развиваться и обучаться смежным или актуальным в команде навыкам и знаниям. Кроме

того, в процессе работ разработчики начнут разбираться в специализациях своих коллег, что позволит снизить так называемый «фактор автобуса проекта» (bus factor) – мера сосредоточения знаний среди отдельных членов команды. Фактор отражает количество разработчиков, в случае потери которых (например, разработчика сбил автобус), проектная деятельность будет существенно затруднена или приостановлена.

3. постоянная – команда разработки должна эффективно решать поставленные перед ней задачи, сформировать такую команду – отдельная и очень сложная задача, которая основана на взаимноличностных отношениях в коллективе. Поэтому эффективно показавшие себя команды разработки не расформировываются после каждого проекта, а сохраняются и переносятся на другой.

- Коллективная ответственность Команды Разработки за создание Инкремента

Говоря о рекомендованном размере команды разработки, необходимо понимать, что для выполнения работы в течение **спринта** весь процесс разработки должен оставаться гибким и продуктивным. В численность команды не включается скрам-мастер, а владелец продукта учитывается в том случае, если он совмещает роль разработчика и выполняет работу из бэклога. Команды, состоящие из 2-3 человек располагают меньшим количеством взаимодействий, знаний и навыков, кроме того, производительность такой команды будет низка. С другой стороны, в командах численностью более 10 специалистов могут возникнуть трудности с коммуникациями и координированием работы. Рекомендованная численность команды - 7 (+/- 2) разработчиков.

Скрам-мастер

Скрам-мастер – специалист в области применения скрама, в чью ответственность входит контроль за процессом функционирования скрама, соблюдение всеми членами команды и заказчика основных правил и практик фреймворка. Как правило, скрам-мастер – это специалист в области организации и менеджмента, он может быть далеко от процессов непосредственной разработки программного обеспечения, но мастерски разбирается в руководстве этим процессом.

Для скрам-команды скрам-мастер выступает в качестве гуру-скрама, идейного лидера, вдохновителя. Он способствует пониманию наиболее полезных взаимодействий в скрам-команде, кроме того, он выступает в роли конфликт-менеджера и организатора внутренней командной культуры. В различных организациях он также выполняет роли коуча, наставника, тренера и вдохновителя развития эффективной команды.

В Таблица 3 представлены различные обязанности скрам-мастера.

Таблица 3

Компетенции скрам-мастера

Для владельца продукта	Для команды	Для организации
Помогает найти наиболее эффективные техники управления бэклогом продукта	Помогает произвести agile-трансформацию процесса разработки	Помогает произвести agile-трансформацию предприятия
Объясняет процесс планирования продукта в эмпирической среде	Устраняет конфликты, препятствия, мешающие прогрессу работы команды	Планирует внедрение скрама в рамках организации
Помогает приоритизировать бэклог продукта	Способствует усвоению и применению принципов и практик гибкой разработки	Консультирует и помогает заинтересованным лицам на этапе внедрения скрама
Способствует пониманию и применению основ и практик гибкой разработки	Помогает в управлении бэклогом спринта команде	Помогает «защитить» перед руководством и ключевыми лицами идеи agile
Помогает организации процесса групповой работы	Помогает организации процесса групповой работы и выстраивает коммуникации	Сотрудничает с другмим скрам-мастерами предприятия для повышения эффективности скрама в рамках всей организации
Проводит тренинги для владельцев продукта	Проводит тренинги для команды разработки	Проводит тренинги для сотрудников компании

Сам процесс разработки с использованием скрама можно представить следующим образом (Рисунок 1-4):

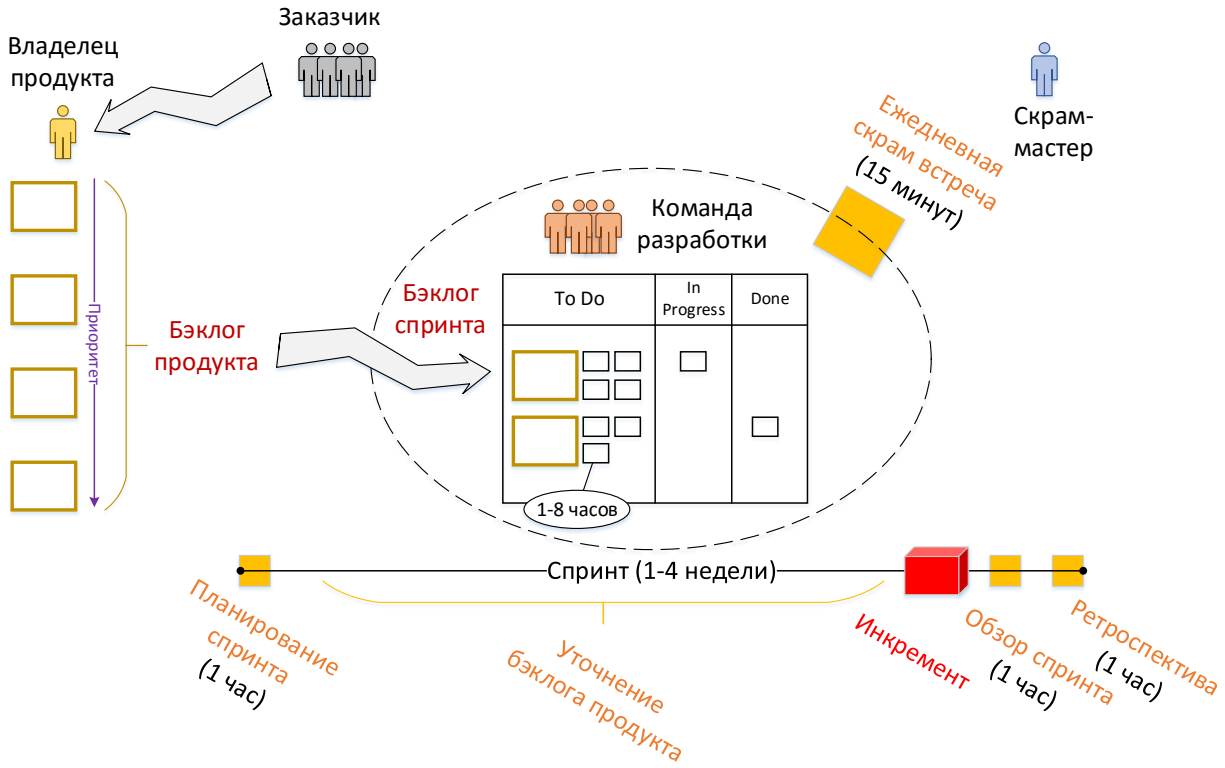

Рисунок 1-4 – **Модель процесса разработки с использованием скрама**

1. Владелец продукта на основе требований от заказчика формирует упорядоченный по приоритету бэклог продукта;
2. Далее происходит планирование спринта: скрам-команда формирует бэклог спринта, декомпозирует элементы бэклога на конкретные задачи, каждая из которых, как правило, не должна занимать более 8 часов На этой стадии формируется скрам-доска, состоящая из трех стадий:
   - To Do (сделать);
   - In Progress (В работе);
   - Done (Сделано).

   Доска позволяет контролировать и координировать процесс работы;
3. Начинается спринт – непосредственная разработка. Задачи берутся из столбца «To Do» и переносятся в столбец «In Progress», при завершении переходят в столбец «Done». Кроме того, в начале каждого рабочего дня проводится «ежедневная скрам встреча», на которой скрам-команда делится своими результатами, возникшими проблемами, что позволяет синхронизировать общие действия. Параллельно процессу работ происходит обсуждение и планирование будущего спринта;
4. В конечной стадии спринта формируется инкремент – продукт спринта, соответствующий поставленным целям. Назначается «обзор спринта», на котором

присутствует вся команда, заказчик и пользователи. Проводится совместно обсуждение полученных результатов;

5. В завершении спринта проводится «ретроспектива спринта», внутренняя встреча скрам-команды, на которой обсуждаются результаты спринта, рабочий процесс, внутренние проблемы. Это делается для улучшения процесса командной работы и повышения эффективности скрам-команды.

6. Kanban – канбан

Если скрам – это революционное изменение процессов предприятия, то канбан – эволюционное.

Канбан – фреймворк, который содержит несколько правил и принципов и предлагает эволюционный переход от привычного в компании образа мышления к agile. Канбан часто сравнивают с водой — он обтекает текущую структуру и иерархию компании, и очень медленно начинает их менять. При внедрении канбана не требуется радикальных изменений процессов компании, что делает этот подход весьма востребованным.

Сотрудникам предприятия не нужно прикладывать много усилий, чтобы начать работать по канбану — при переходе нет реорганизаций, на первых порах сохраняются привычные роли. Всё проходит в комфортном темпе и не вызывает проблем у команд.

Правила и принципы канбана могут вводиться постепенно и поочередно. Новое не добавляется, пока первые изменения не стали привычными для большинства сотрудников.

Из-за эволюционного подхода, позволяющего модернизировать имеющиеся в компании процессы, канбан позволяет щадить старый порядок и культуру работы, поэтому, в отличие от скрама, не требуется большой работы по поводу обоснования и разъяснения причин трансформации бизнеса.

Смысл использования канбана можно описать следующим образом (Рисунок 1-5):

- Представим имеющийся в компании процесс в качестве «трубы», состоящей из составных частей, соответствующих стадиям этого процесса;
- Определим «мощность» каждой стадии: например, известно, что за одну рабочую неделю аналитики успевают решить 2 задачи, проектировщики 4, разработчики 1-2, а тестировщики 6.
- Задача канбана – оптимизировать имеющуюся систему для получения максимально эффективного выпуска без простоя работы. Так, для рассматриваемого примера, проектировщики и тестировщики могут подключать свои силы к процессу разработки, тем самым увеличивая мощность «отстающих» стадий.

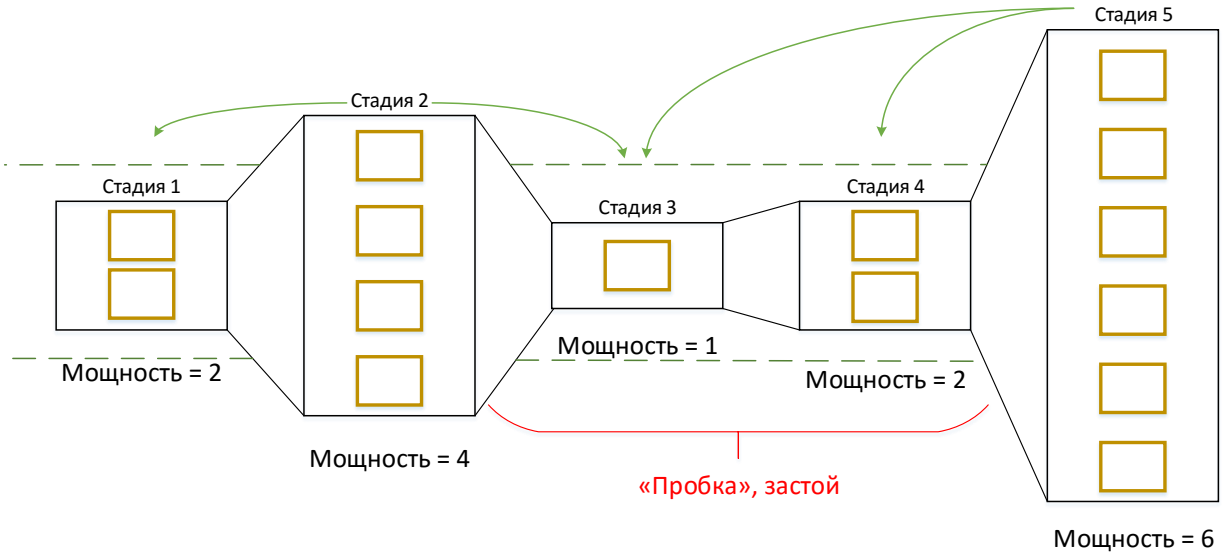

Рисунок 1-5 – **Схема работы по канбану**

Для построения работы по канбану необходимо реализовать на предприятии следующие правила:

1. **Визуализация при помощи «канбан-доски»**

Основная agile-практика – визуализация всей работы при помощи доски, занимает основополагающее место в канбане. На Рисунок 1-6 представлен пример реализации канбан-доски.

Бэклог	Стадия 1		Стадия 2		Стадия 3		Стадия 4		Стадия 5	
	In Progress	Done	In Progress	Done	In Progress	Done	In Progress	Done	In Progress	Done

Рисунок 1-6 – **Канбан-доска**

Она похожа на скрам-доску, но имеет ряд определяющих отличий:

- Доска разбита на стадии разработки, каждая из которых делится на работу «In Progress» - в процессе и «Done» - сделано;
- Бэклог определяется владельцем продукта, он может в любой момент поменять приоритет элементов бэклога, что позволяет «вмешиваться» в рабочий процесс и продвигать наиболее важные задачи;
- Разработчики всегда «вытягивают» самую высокую по приоритету задачу, как только задача выполнена – берется следующая;
- Все задачи должны быть отражены на доске.

После создания первой доски будет выявлено как много работы выполняется параллельно и сколько задач уже выполнено, но не взято в работу на следующей стадии.

Это одна из причин, почему задачи растягиваются: силы тратятся не на продуктивную работу, а на переключение между задачами и застой.

2. Необходимо ограничить количество одновременно выполняемой работы

Использование канбана подразумевает ограничение количества одновременно выполняемой работы на каждой стадии. Это позволит повысить эффективность команды и увеличить скорость прохождения задачи по каждой стадии до конечного статуса «Done».

Необходимо проанализировать, какое количество задач на каждой стадии можно выполнять одновременно, после чего зафиксировать этот лимит.

Ограничение указывается для каждой стадии для того, чтобы команда могла в любой момент времени получить актуальную информацию о статусе имеющихся работ (Рисунок 1-7).

Рисунок 1-7- **Канбан-доска с ограничениями**

3. Необходимо управлять потоком имеющихся задач

Канбан-доска позволяет наглядно демонстрировать скорость продвижения задач и равномерность загрузки тех или иных специалистов, кроме того, доска позволяет мониторить возникновение каких-любо проблем или задержек.

Каждый специалист должен следить за текущей загрузкой: если чувствуется спад ритмичности, значит, скорее всего, на каком-то этапе не хватает мощностей, поэтому имеет смысл предложить свою помощь. Помощь не подразумевает то, что незагруженные специалисты будут делать чужую работу: члены коллектива сами определяют насколько сильно они готовы расширять свою зону ответственности.

Сотрудники должны осознавать ответственность и, если возникает необходимость, обязательно сообщать о проблемах и просить помощи. От эффективности и скорости отдельного этапа напрямую зависит результат общей работы.

4. Прозрачные договорённости и правила

Необходимо, чтобы каждый сотрудник знал и понимал правила работы команды. Хорошей практикой считается письменная фиксация всех правил, комментариев, нюансов рабочего процесса каждой стадии. Описанные правила можно повесить прямо на канбан-доску, таким образом, они будут доступны в любой момент времени каждому сотруднику.

Между сотрудниками разных стадий могут быть определенные договоренности, их также необходимо фиксировать и публиковать на канбан-доску для того, чтобы любой сотрудник мог с ними ознакомиться.

Так как процесс зависит от слаженной работы всей команды, то все правила и договоренности должны быть известны каждому ее члену.

5. Регулярный анализ деятельности команды

Постоянный анализ результатов работы команды – обязательное требование Канбана. Мониторинг деятельности команды позволяет контролировать направление рабочего процесса и держаться установленных сроков.

Каких-то конкретных и специфичных ограничений или правил по частоте и формату встреч команды нет – подходит большинство используемых аджайл-практик, но эффективнее всего себя показывают использование практик ежедневных, еженедельных и ежемесячных планерок.

Ежедневные встречи не занимают много времени и позволяют координировать действия команды, обсуждать и решать текущие проблемы и вопросы внутри команды.

Еженедельные встречи позволяют анализировать результат рабочей недели, на них присутствует вышестоящее начальство, обсуждаются идеи по поводу повышения скорости и эффективности работы.

В ежемесячной встрече участвуют все команды компании и руководство. Обсуждаются финансовые результаты, решеные задачи, комментируется деятельность каждой команды, чтобы все сотрудники могли знать о деятельности компании.

Важно, чтобы был контакт с каждым членом команды, и все имели понятие о текущей деятельности своих коллег.

6. Эволюционное развитие процессов на основе командной инициативы и экспериментов

Особенность работы по канбану заключается в том, что канбан-команды всегда находятся в процессе поиска идеальной рабочей системы, в которой скорость выполнения работы протекает максимально быстро.

Это возможно только тогда, когда команды имеют некоторую самостоятельность и свободу принятия решений по поводу процесса работы. Основой эволюционирования процессов являются инициативность и мотивированность каждого члена команды: необходимо всегда пробовать что-то новое, экспериментировать и выбирать лучшие практики.

Каждый член команды должен предлагать и обосновывать перед командой изменения, которые, по его мнению, могут привести к повышению эффективности

процесса. Даже если команда не удовлетворена обоснованием, рекомендуется проводить эксперименты, которые не скажутся на рабочем процессе, но могут быть источниками новых технологических открытий.

Выводы по главе

Высокая активность в области разработки ПО стала следствием появления большого количества частных методологий. Появилось направление бизнеса, связанное с консалтингом в области подходов к разработке ПО, компании рассматривают применяемый у них подход как некий нематериальный актив, имеющий высокую степень влияния на проектную деятельность, что является показателем важности темы выбора подходящей под критерии IT-проектов компании методологии. Например, в работе Павленко Е.П [15] предпринимается попытка выбора между гибкой методологией разработки ПО и классической (каскадная, инкрементальная, спиральная модели) для работы страховой компании. Автор приходит к выводу, что гибкая модель отвечает специфике бизнеса лучше, но исследование опирается только на субъективные ощущения автора и не подкреплено иными видами анализа.

Выбор того или иного подхода остается большой проблемой, как правило, для понимания возможностей применения той или иной методологии требуется весьма серьезный опыт работы в области разработки ПО. Кроме того, руководители не рискуют варьировать несколькими методами в рамках проектной деятельности, как правило, они используют один хорошо им известный. Сложилось мнение, что существует большая степень неопределенности при использовании новой технологии разработки, поэтому руководство решает применять уже известную им методологию вместо наиболее гибкой или оптимальной в конкретных условиях, обосновывая это снижением рисков.

Эта проблема является причиной многих исследований, цель которых - обоснование преимущества или недостатков какой-то частной методологии. Например, в исследовании «Agile software development methods: Review and analysis» [23] производится серьезный анализ существующий гибких методологий разработки ПО. Но это исследование не позволяет определить ситуации, в которых наилучшим образом найдет свое применение какой-то из методов. Исследование частных методологий и их поверхностное сравнение между собой являются результатом не всегда достаточно обоснованных выводов, не подкрепленных практическим опытом применения. Поэтому следующая задача исследования – провести детальный обзор и сравнительный анализ рассмотренных в этой главе методологий.

Глава 2 Существующий инструментарий выбора методологии разработки ПО

2.1 Предпосылки использования методологий разработки ПО

Для понимания причин возникновения различных методологий разработки программного обеспечения обратимся к концепции Кеневина (Cynefin Framework) [27], предложенной Дэйвом Сноуденом, экспертом, возглавлявшим принадлежащий IBM институт управления знаниями, затем их же центр изучения приложений теории сложных систем к развитию организаций. В 2003 году Сноуден предложил концепцию, основной смысл которой заключается в том, что весь мир и все процессы можно рассматривать как системы четырех типов (Рисунок 2-1). Каждая из четырех систем, по заключению Сноудена, соответствует определенному способу восприятия и понимания проблемы или ситуации - от того, к какой области мы отнесем проблему, будет зависеть стратегия нашего мышления - как мы будем воспринимать ситуацию и какие выводы делать.

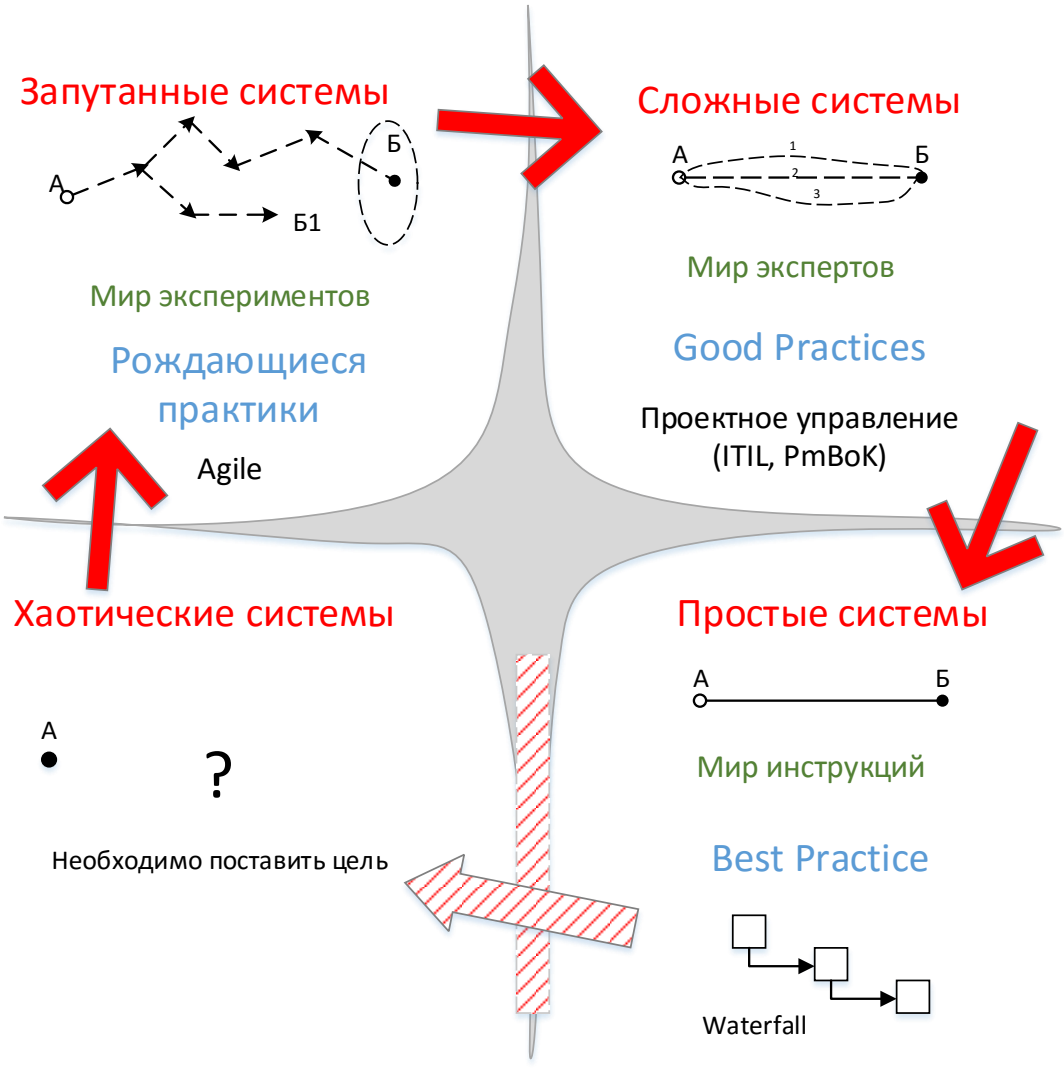

Рисунок 2-1**. Концепция Кеневин (Cynefin Framework)**

Простые системы

Это область простого порядка, в которой у событий и явлений есть явные и однозначные причины, и они всегда приводят к определенным и неизменным следствиям. Благодаря этому можно точно предсказывать результаты того или иного действия, делать точные прогнозы и на их основе точно "вычислять" лучший шаг.

Простые системы – это мир инструкций. Все, для чего могут быть написаны точные и относительно простые инструкции, лежит в этой области. Например, к простым системам можно отнести сборку мебели при помощи инструкции.

Сложные системы

Область сложного порядка. Здесь, как и в области простого порядка, происходящее обусловлено твердыми причинами и следствиями, но связи между ними запутаны. В принципе, мы можем привести происходящее в этой области к области простого порядка, но не всегда для этого достаточно ресурсов и времени. Поэтому в этой области хорошим вариантом является поиск эксперта, который бы мог эффективно анализировать переплетение причин и следствий и предоставлять нам рекомендации.

Сложные системы – мир экспертов, определяющих способ достижения цели. Все, что может быть проанализировано логически и математически (в том числе и изощренными, сложными методами), лежит в этой области. Так, к этой области можно отнести процесс постройки дома. Разные эксперты дадут разные мнения по поводу того, каким образом дом построить «правильно».

Запутанные системы

Запутанные системы - область сложных нелинейных систем. Сноуден определяет эту дисциплину как изучение конфигураций, возникающих в результате взаимодействия большого количества объектов: в этих взаимодействиях имеются свои причины и следствия, но большое число объектов-участников и еще большее число взаимодействий между ними не позволяют для понимания ситуации использовать классификацию или анализ. Возникающие конфигурации кажутся объяснимыми только ретроспективно – существует возможность воспринимать и объяснять их, но не предсказывать. Повторение конфигураций возможно, но на них нельзя опираться как на прочные закономерности, потому что конфигурации могут исчезнуть или измениться в любой момент - от нас скрыты элементарные взаимодействия, и мы не можем этот момент предвидеть.

Запутанные системы – мир экспериментов, подхода «проб и ошибок».

Сноуден подчеркивает, что для успешного восприятия и понимания в этой области требуется множественность точек зрения на ситуацию. Вместо того, чтобы спешить с выводами, "хватаясь" за первую знакомую конфигурацию, следует спокойно и

сосредоточенно наблюдать, открыв сознание для новых альтернативных пониманий ситуации.

Ситуации, когда слишком много больших и малых вещей влияет на дело, когда нет "основных причин", а следствия неоднозначны, когда ситуацию можно объяснить, но не предвидеть - все они лежат в этой области. Примером может служить маркетинговая ситуация на каком-нибудь высоко-конкурентном рынке, динамика курса валют или формирование урагана над Атлантикой.

Хаотические системы

В этой области связи между причинами и следствиями, кажется, отсутствуют. Ситуация хаотична, турбулентна. Достоверно отнести происходящее к какой-либо знакомой категории невозможно. Анализу ситуация не поддается. Неопределенность ситуаций в области хаоса создает впечатление их "неуютности", "опасности". Даже если видеть в этом хаосе корни будущего порядка, требуется смелость и решительность, чтобы начать действовать в таких условиях. Вероятно, единственная разумная последовательность действий в этой области:

- Действовать быстро и решительно против хаоса и неопределенности;
- Наблюдать немедленную реакцию на свои действия;
- Корректировать свои действия и энергично действовать дальше;
- Эта тактика должна вывести нас, в конце концов, в одну из трех остальных, более "комфортных" областей.

Очень важно, что именно область хаоса - источник инноваций и коренных перемен. Именно тут различные уровни систем переплетаются в одном узле, так что возможен переход с уровня на уровень. Поэтому в поисках изменений иногда целесообразно погружаться в область хаоса сознательно, хотя это не может обходиться без риска.

Пример ситуации в области хаоса - крах советской экономики в начале 90-х годов.

Как показано на Рисунок 2-1, при определенных условиях системы могут менять свое состояние. Например, после проведения детального анализа и формирования конкретных целей и задач, хаотическая система может стать запутанной. Таким же образом все остальные системы могут стать более определенными.

С другой стороны, при неверно построенных процессах или ошибочной постановке цели и задач, система может стать более сложной. Более того, иногда простая система может стать хаотической. Например, опытные сотрудники компании могут прекрасно разбираться во внутренних процессах компании, выполнять все свои должностные обязанности и взаимодействовать с коллегами и начальством. Для них это простая система. Но, если вдруг у компании сменится собственник или руководство, опытные сотрудники

уже не будут знать, что именно от них ожидает новое начальство, какие новые корпоративные правила будут действовать в организации.

Концепция Кеневин прекрасно подходит и описывает область разработки программного обеспечения.

Все проекты по разработке могут быть представлены как одна из предложенных систем либо рассмотрены через их проекции, например, на Рисунок 2-2 представлено совмещение концепции Кеневин и модели жизненного цикла программного обеспечения.

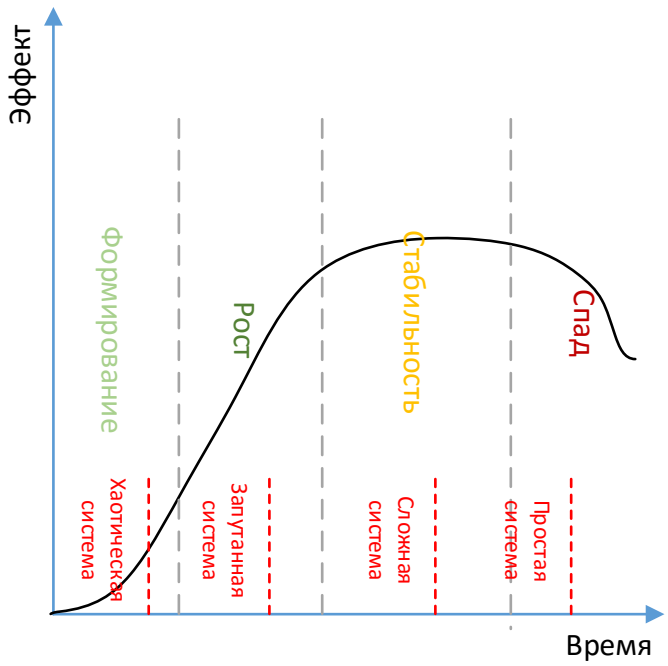

Рисунок 2-2. **Концепция Кеневин и модель жизненного цикла ПО**

Как правило, на стадии формирования проекта по разработки программного обеспечения специалисты сталкиваются с проблемой непонимания либо того, как делать ту или иную задачу, либо того, что именно нужно делать. Это очень похоже на хаотическую систему.

Через некоторое время, когда проект находится на стадии активного роста, когда специалисты уже разбираются в предметной области и представляют себе результат проекта - система становится запутанной.

Когда проект переходит в стадию стабильности, когда среди специалистов появляются эксперты, когда появляется понимание результата и того, что для его достижения необходимо предпринять – система перестает быть запутанной и становится сложной.

Наконец, когда проектная деятельность подходит к завершению, либо продукт-результат проектной деятельности теряет актуальность, когда процессы становятся рутиной – система становится простой.

Кроме того, предложенную концепцию можно представить с позиции понимания объекта и процесса разработки так, как показано на Рисунок 2-3.

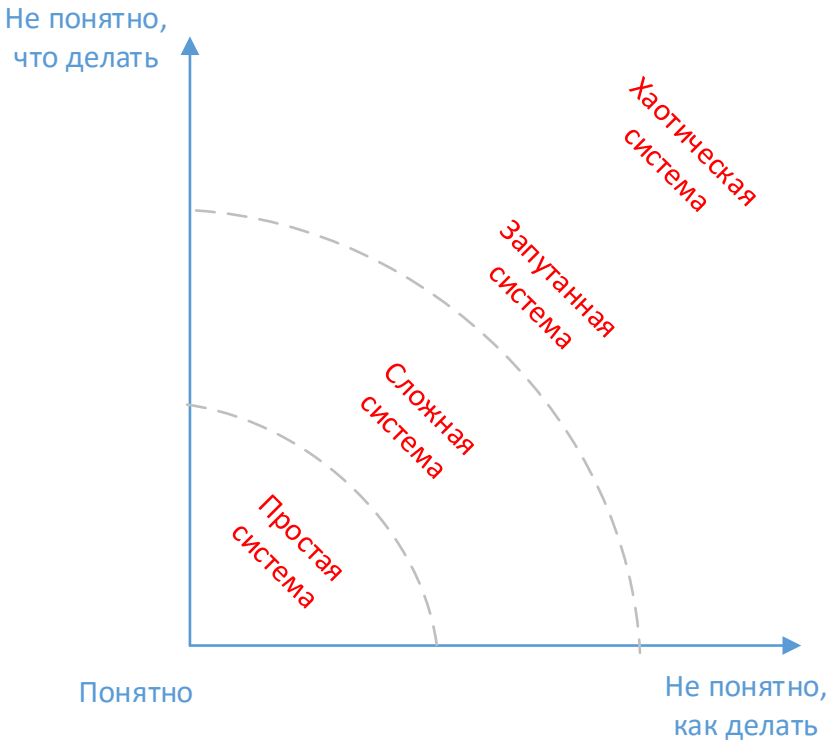

Рисунок 2-3. **Концепция Кеневин в проекции понимания процессов**

Когда специалистом ясно что нужно делать и как это нужно делать – перед нами простой проект, который можно соотнести с простой системой.

По мере увеличения неопределенности по поводу того «что» или «как» нужно делать – проект становится сложнее и соответственно усложняется система.

С концепцией Кеневин очевидным образом сопоставляются методологии разработки программного обеспечения (Рисунок 2-1).

Простая система – мир инструкций – полностью подходит под описание модели разработки Waterfall.

Сложная система – мир экспертов – использование накопленных практик, например, проектное управление с использованием ITIL.

Запутанная система – мир экспериментов – прекрасно подходит под описание гибкой методологии разработки.

В случае с хаотической системой требуется провести аналитическую работу для того, что перевести проект в систему с большей мерой определенности.

Но возникает логичный вопрос: каким образом определить к какой системе можно отнести тот или иной проект? А кроме того, если сопоставить проект с какой-то конкретной системой, означает ли это, что нужно однозначно использовать какую-то определенную методологию разработки [8]?

Современное развитие области методологий разработки программного обеспечения определяет, что однозначного ответа на вышестоящие вопросы нет. Нельзя сказать, что определенная методология всегда будет решать задачу построения процесса разработки определенного типа проектов лучше остальных. Так или иначе, необходимо учитывать частные характеристики и риски каждого проекта, причем для различных команд или компаний эти характеристики и риски могут значительно отличаться.

Интуитивно-эмпирический подход, распространенный среди руководителей проектных команд и проектной деятельности, влечет за собой большие риски того, что будет неправильно поставлен весь процесс разработки, что может стоить больших временных и экономических потерь или разрыва сотрудничества с заказчиком. Это обуславливает важность задачи поиска инструмента коллективного и прозрачного принятия решения по поводу использования той или иной методологии.

2.2 Концептуальное описание модели выбора методологии разработки программного обеспечения

Предлагаемая модель выбора методологии разработки программного обеспечения построена на концепции инструмента Оценки рисков (RAF - Risk Assessment Framework) для подхода Планирования сервисов уровня предприятия (Enterprise Services Planning). RAF был предложен в 2016 году специалистами из сообщества Kanban Community, по причине того, что существующий инструмент Weighted Shortest Job First (WSJF -Более Ценная и Короткая Работа Сначала) не всегда показывал свою состоятельность, так как в большинстве предприятий выявляется отсутствие корреляции между оценкой проекта и длительностью его исполнения [38].

Модель выбора представляет собой систему координат, в которой параметры проекта являются осями. Количество осей может варьироваться в зависимости от характеристик проекта и целесообразности использования тех или иных его параметров.

Все оси строятся из единой точки отсчета – точки начала координат.

Для каждого параметра необходимо выбрать характеристики, которые он может иметь и которые существенно влияют на процесс проектной деятельности. Характеристики располагаются на осях таким образом, что чем ближе к точке начала координат – тем ниже риски, связанные с таким проектом, чем дальше от точки начала координат– тем риски выше.

Шаг 1. При согласовании количества параметров и их характеристик необходимо нанести их на диаграмму (Рисунок 2-4).

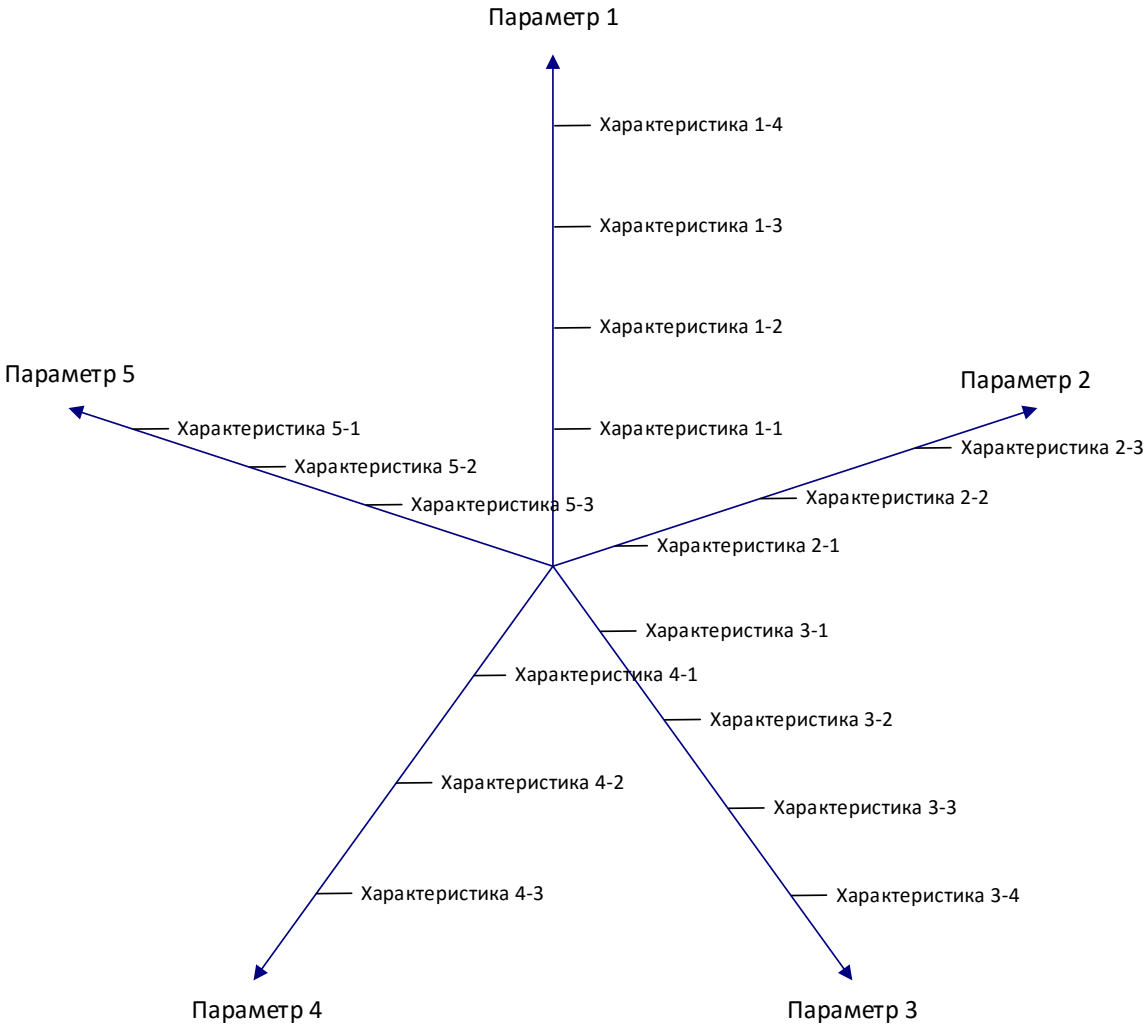

Рисунок 2-4. **Диаграмма параметров проекта и их характеристик**

Источник: собственная разработка

Шаг 2. Далее необходимо на основе имеющихся знаний и накопленного опыта применения той или иной методологии нанести на диаграмму профили методологий разработки ПО. Требуется разложить каждую методологию по имеющимся параметрам и выбрать те характеристики, которые ей соответствуют или по отношению к которым применение методологии наиболее эффективно и востребовано (Рисунок 2-5).

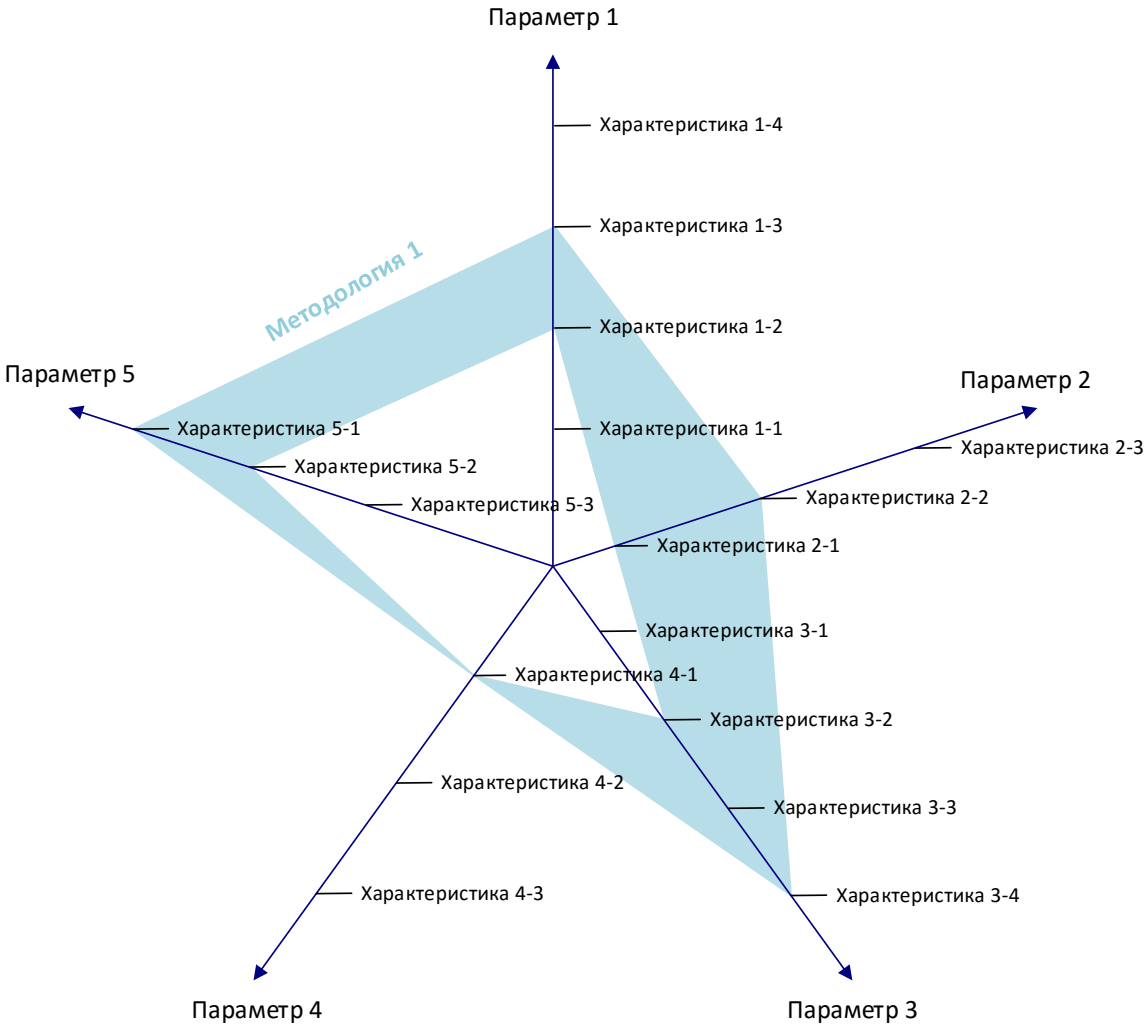

Рисунок 2-5. **Профиль методологии разработки на диаграмме параметров проекта**

Источник: собственная разработка

Полученная диаграмма с профилем методологии является промежуточным результатом модели. Он позволяет наглядно отображать возможности применения той или иной методологии в рамках имеющихся параметров.

Шаг 3. Для следующего шага необходимо проанализировать имеющиеся проекты, соотнести их с имеющимися параметрами и их характеристиками и нанести профили проектов на диаграмму (Рисунок 2-7).

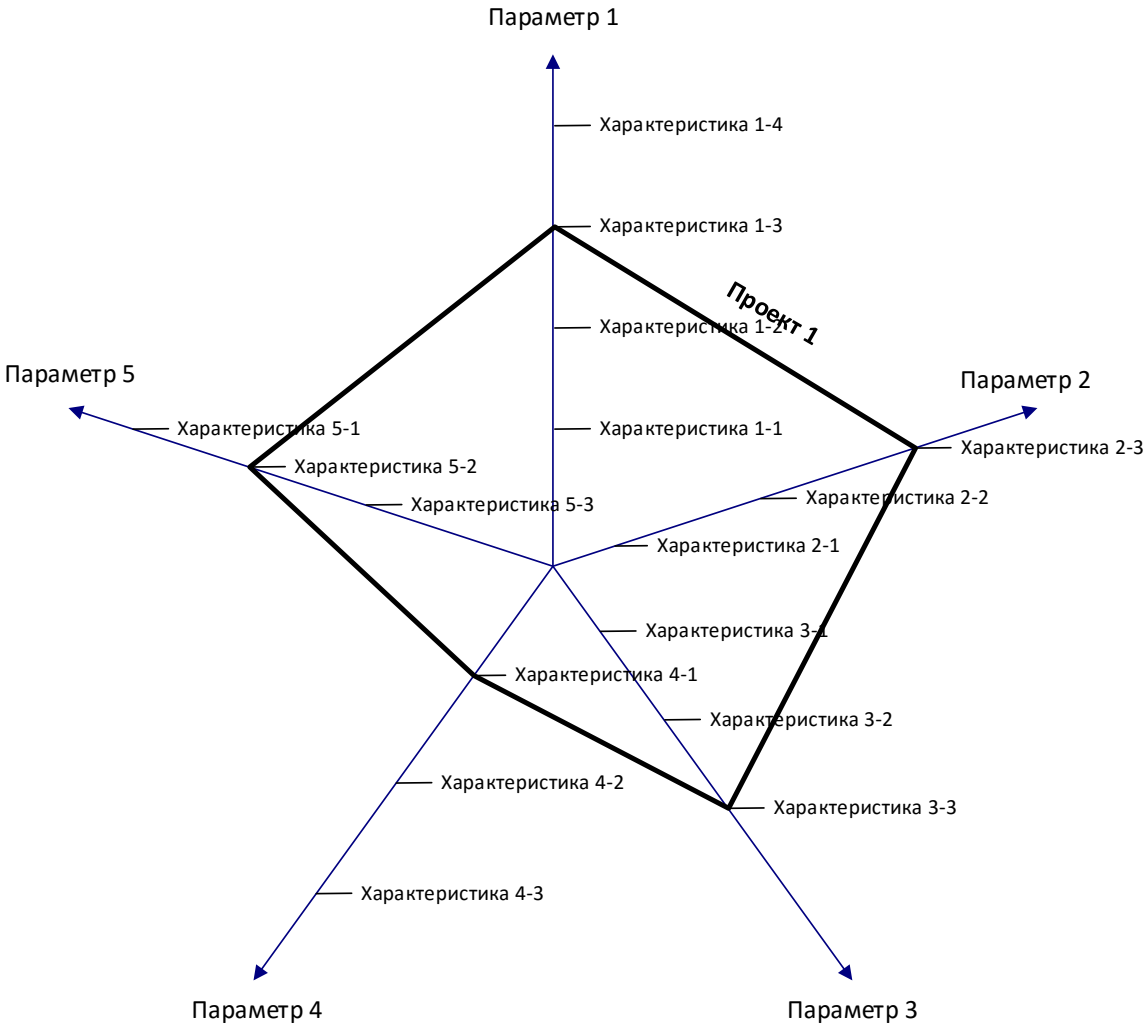

Рисунок 2-6. **Профили проекта и методологии разработки на диаграмме параметров проекта**

Источник: собственная разработка

Диаграмма позволяет однозначно определить параметры конкретного проекта по рассматриваемым параметрам.

Шаг 4. Следующий шаг – соотнесение диаграммы профиля проектов с диаграммой профиля методологии разработки ПО (Рисунок 2-7).

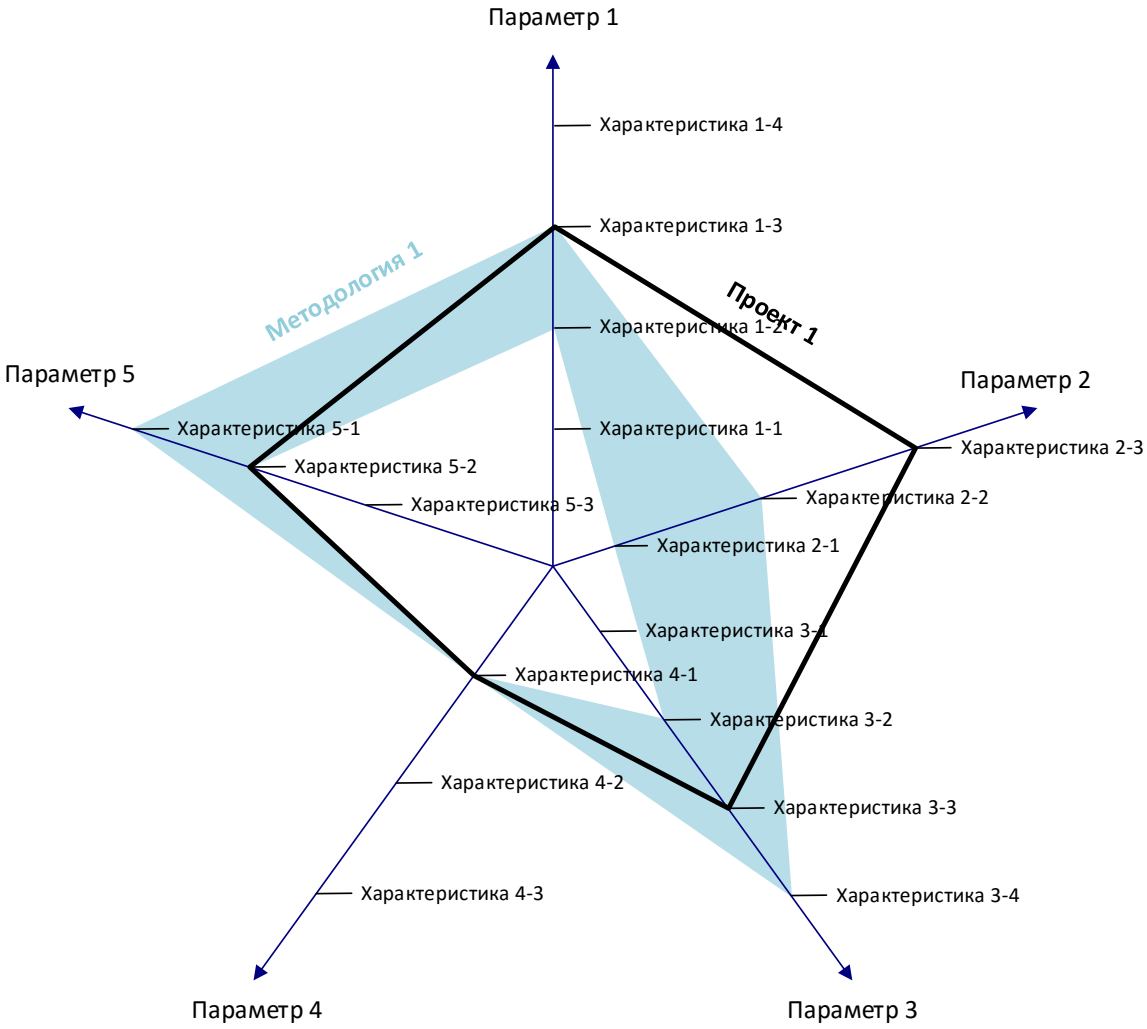

Рисунок 2-7. **Модель выбора методологии разработки программного обеспечения**

Источник: собственная разработка

Диаграмма наглядно демонстрирует, что «Проект 1» соответствует «Методологии 1» по всем параметрам, кроме «Параметра 2».

Модель позволяет обсудить имеющиеся риски и принять решение о выборе той или иной методологии разработки.

В случае, если один из параметров становится неактуальным или его использование нецелесообразно – его можно исключить. Если возникает сомнение выбора значений параметров или их расположение на осях – то модель также позволяет менять их количество и расположение на более показательные.

Выводы по главе

Исследования в области процессов разработки программного обеспечения и их активное применение на практике повлекли за собой появление большого количества специфических методологий разработки ПО. Но, на данный момент, вопрос выбора той или иной методологии в зависимости от типа проекта остается открытым: эксперты признают отсутствие возможности формализовать подход выбора методологии до какого-то принципиального правила.

Мной предложена модель, позволяющая:

1. Провести анализ проектной деятельности предприятия и выделить наиболее критичные параметры проектов и их рисковые характеристики путем создания «Диаграммы параметров проекта и их характеристик» (Рисунок 2-4);
2. Рассмотреть и соотнести используемые в проектной работе методологии разработки программного обеспечения с выделенными параметрами проектов, в результате чего получить «Диаграмму профиля методологии разработки ПО» (Рисунок 2-5);
3. Квалифицировать конкретный проект компании в рамках исследуемых параметров и тем самым построить «Диаграмму профиля проекта»;
4. Соотнести диаграммы из двух предыдущих пунктов, тем самым создав «Модель выбора методологии разработки программного обеспечения», которая позволяет определить возможные риски при реализации проекта с использованием той или иной методологии.

Предложенная модель позволяет:

- анализировать выбор методологии разработки ПО в зависимости от параметров проекта, что позволяет принимать более обоснованные решения, подкрепленные специализированной моделью;
- наглядно демонстрировать возможные при выборе той или иной методологии риски, что позволяет их вовремя минимизировать или нивелировать;
- гибко менять возможные параметры проектов в зависимости от имеющихся условий, не уменьшая состоятельности модели.

Глава 3 Практическое применение модели выбора методологии разработки программного обеспечения

3.1 Выделение параметров модели выбора и их значений

Первый шаг модели – построение диаграммы параметров проекта.

Для проведения анализа возможных параметров и их значений необходимо собрать экспертов, участвующих в процессе проектной деятельности. Лучше всего для этого подходит очный «мозговой штурм» команды состоящей из следующих экспертов:

- Владелец продукта, менеджер продукта, менеджер проекта – специалист, отвечающий за проект или продукт, являющийся результатом проекта. Как правило, этот специалист является лицом, принимающим решение. Эта категория экспертов обладает обширной информацией о самом проекте/продукте и связанными с ним рисками;
- Бизнес-аналитик – специалист, отвечающий за выявление и формирование требований при непосредственном общении с заказчиком. Эта группа экспертов досконально изучает и глубоко погружается в предметную область и саму специфику проекта/продукта, поэтому имеет знания о всех возможных «подводных камнях»;
- Разработчик, тестировщик– специалист, отвечающий непосредственно за техническую реализацию проекта, за его функциональные возможности и соответствие результата ожиданиям заказчика.

Чем больше компетентных специалистов будет участвовать во встрече, тем объективнее окажется результат анализа, ведь параметры и их значения необходимо выбирать не большинством голосов, а полным согласием насчет того или иного параметра. К примеру, все эксперты пришли к определенному мнению насчет какого-то параметра, но один высказался против. Может оказаться, что этот специалист погружен в определенную область проекта глубже и знает что-то такое, что не учли при анализе остальные эксперты. После того, как специалист обоснует свою позицию, обсуждение вопроса продолжается: остальные эксперты могут принять обоснование и поменять свою позицию, либо могут убедить специалиста в том, что их позиция учитывает его сомнения.

В экспертную группу компании Qlever Solutions были включены следующие специалисты:

- Технический директор компании – совмещает в себе роли архитектора и главного разработчика;

- Руководители проектов – отвечает за организационную составляющую проекта, общение с заказчиком, формирование требований и документации, частично разработку;
- Бизнес-аналитики – специалисты, задачей которых является погружение в предметную область заказчика. Они являются ответственными за сбор и разработку требований, разработку и ведение документации, коммуникацию между группой разработки и заказчиком;
- Разработчики – технические специалисты, в чьи задачи входит непосредственно разработка ПО, его тестирование.

На основе накопившегося в компании опыта работы с заказчиками и практик разработки программного обеспечения, экспертная группа выделила следующие специфичные параметры проектов, представленные в Таблица 4. Рассматриваемые параметры оказывают особое влияние на проектный процесс в компании Qlever Solutions, рискованность той или иной характеристики параметра возрастает по мере увеличения значения в столбце «Риск».

Таблица 4.

Параметры проектов компании Qlever Solutions

Параметр	Риск	Характеристика параметра	Комментарий
Технические риски	1	Рутина	Разработка такого ПО является для компании обычной деятельностью, рутиной
	2	Мы уже такое делали	Аналогичное решение уже когда-то разрабатывалось
	3	Делали, но не мы	Приходилось встречаться с подобным решением либо оно уже существует у заказчика
	4	Неизвестное решение	Впервые сталкиваемся с разработкой подобного ПО

Продолжение таблицы 4

Сроки поставки	1	Не срочно	Заказчику не принципиален срок поставки первого прототипа
	2	Фиксированная дата поставки	Срок поставки прототипа фиксирован
	3	Регулярная поставка	Заказчику необходимо регулярно получать обновленные прототипы
	4	Срочно, нужно уже сейчас	Заказчику требуется прототип в самые короткие сроки
Требования	1	Известны и фиксированы	Заказчик точно знает, что он хочет, и требования не могут меняться
	2	Известны, но могут меняться	Заказчик знает, что он хочет, но не исключает некоторые изменения в требованиях в процессе разработки
	3	Не известны, будут сформированы в процессе	Заказчик имеет представление о конечном результате, но пока что не может точно сформулировать требования
Бюджет	1	Time & Materials	Оплата по факту выполнения работ: заказчику предоставляется расчет потраченного на разработку времени и прочих ресурсов
	2	Рамочный договор	Заключается договор на определенную сумму, в рамках которой может реализовываться проект
	3	Может быть расширен	Сумма договора фиксирована, но может быть заключен дополнительный договор или расширен текущий
	4	Фиксирован	Сумма договора на разработку фиксирована и не может быть изменена

Продолжение таблицы 4

Параметр	Риск	Характеристика параметра	Комментарий
Вовлеченность заказчика	1	Полная	Заказчик полностью вовлечен в проектный процесс, возможна занятость специалистов заказчика в проектную работу на полный рабочий день
	2	По мере возможностей	Заказчик имеет ограниченную возможность участвовать в проектной работе, только при отсутствии занятости по своим основным должностным обязанностям
	3	Минимальная	Заказчик не заинтересован в участии в проектной работе, полностью занят основными должностными обязанностями
Проектные риски	1	Преобладают контролируемые риски	Преобладают контролируемые проектные риски, их вероятность не высока
	2	Преобладают частично контролируемые	Преобладают частично контролируемые проектные риски
	3	Преобладают неконтролируемые риски	Преобладают неконтролируемые риски, вероятность достаточно высока

Далее полученные параметры и их значения наносятся на диаграмму (Рисунок 3-1).

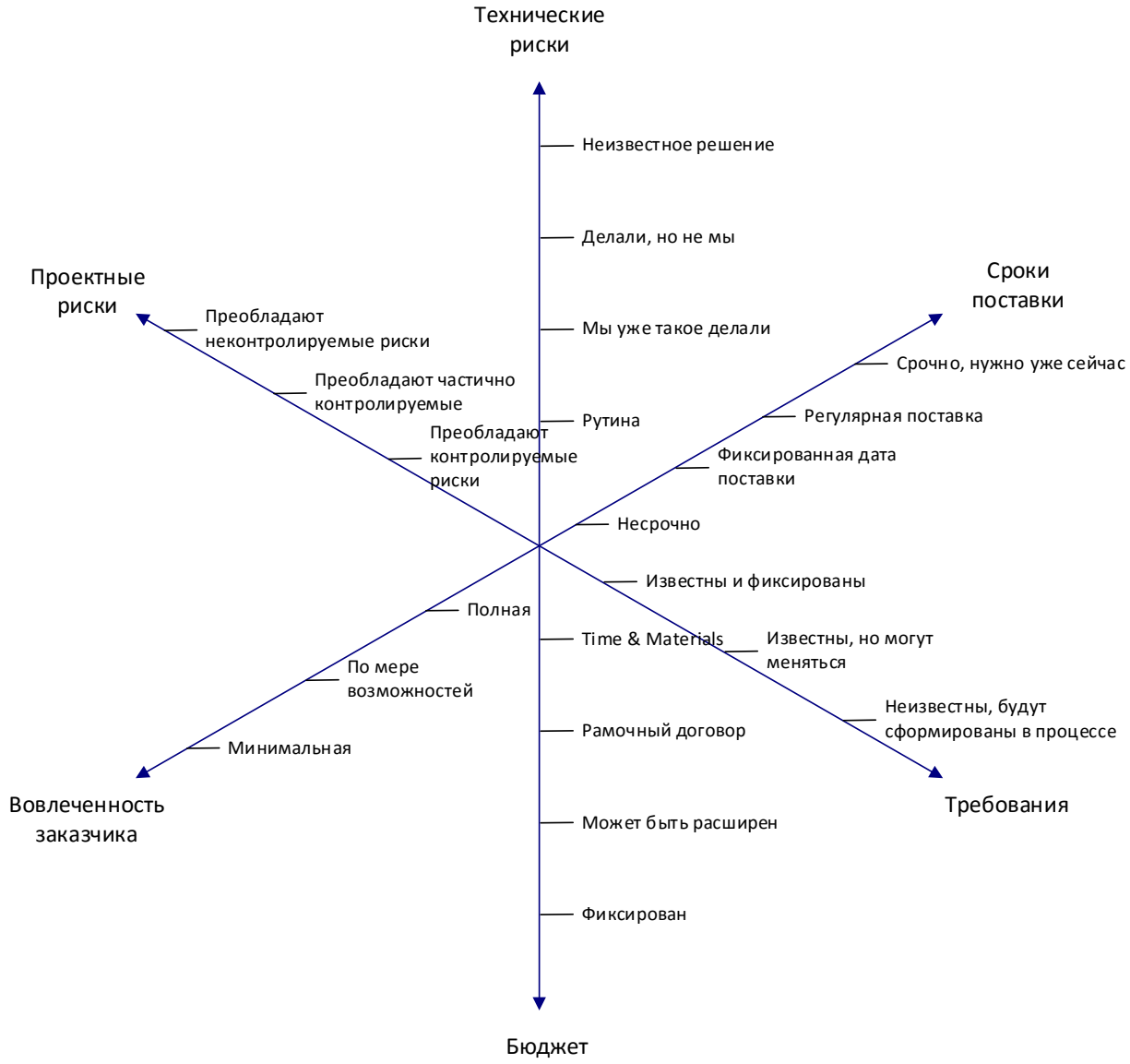

Рисунок 3-1. **Диаграмма параметров проекта для компании Qlever Solutions**

Источник: собственная разработка

В результате чего получается «Диаграмма параметров проекта», используемая для оценки проектов с возможностью их классификации.

3.2 Создание модели профилей методологий разработки ПО на диаграмме параметров проектов

Следующий шаг: необходимо нанести профили используемых в компании методологий разработки программного обеспечения на диаграмму параметров проекта.

В компании сложилась практика использования следующих методологий:

- Каскадная (Waterfall);
- Спиральная (Spiral);
- Scrum;
- Kanban.

Каскадная модель разработки ПО

По сложившейся корпоративной практике каскадная модель используется в небольших проектах (менее полугода), когда заказчик точно знает, что он хочет, а разработчики знают, как это сделать.

В этой методологии анализ и проектирование являются самыми важными процессами, так как допущенная на этих стадиях ошибка будет иметь самую дорогую стоимость.

Профиль каскадной модели, нанесенный на диаграмму параметров проекта представлен на Рисунок 3-2.

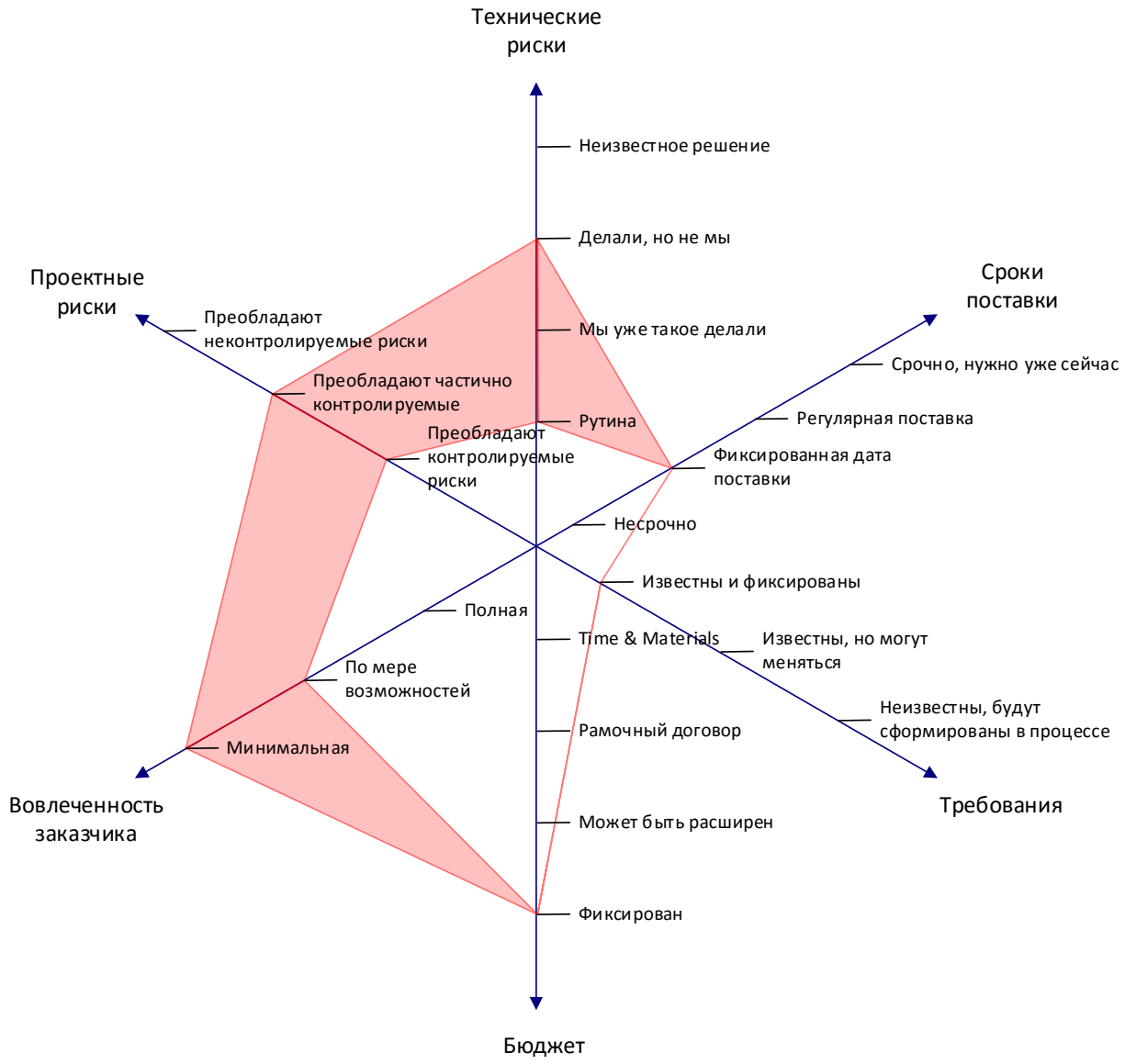

Рисунок 3-2. **Профиль каскадной модели на диаграмме параметров проекта**

Источник: собственная разработка

Спиральная модель разработки ПО

Спиральная модель используется в сложных или экспериментальных проектах, когда нет четкого понимания результата или высокие риски нереализации продукта. Она позволяет детально контролировать проектный процесс и корректировать его по мере выполнения.

Профиль спиральной модели, нанесенный на диаграмму параметров проекта представлен на Рисунок 3-3.

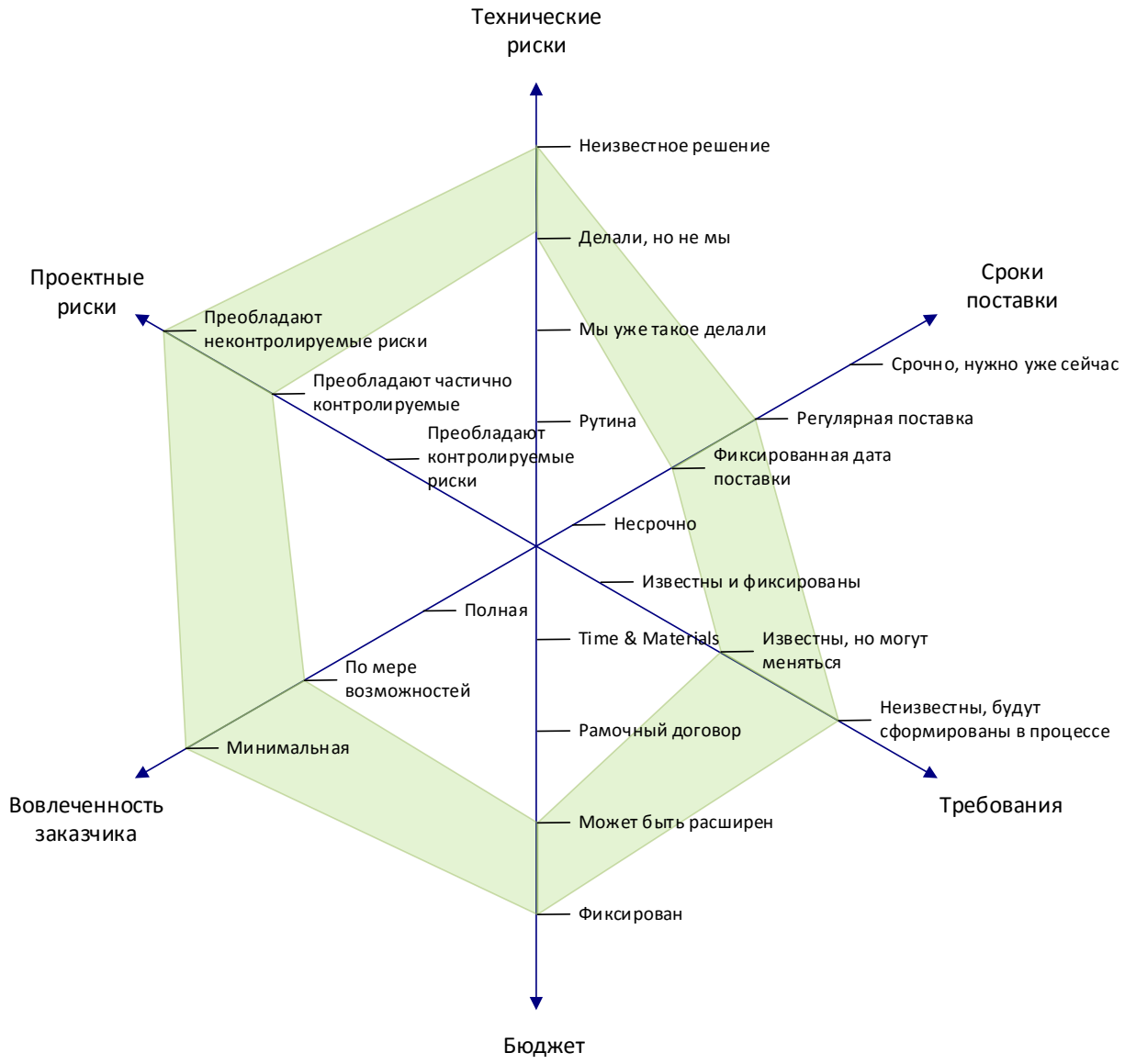

Рисунок 3-3. **Профиль спиральной модели на диаграмме параметров проекта**

Источник: собственная разработке

Методология Scrum

Для использования при работе принципов гибкой разработки, а тем более всех практик, объединенных под методологией Scrum, необходимо несколько условий:

1. Заказчик должен понимать и соглашаться с тем, что он не может влиять на состав бэклога спринта и процесс разработки. Он может только формировать и приоритизировать верхнеуровневый бэклог, но разработчики сами выбирают задачи, которые будут включены в спринт;
2. Заказчик должен понимать, что у проекта могут быть изменены временные и бюджетные границы;

3. Существует постоянная, самоорганизованная скрам-команда, состоящая из скрам-мастера, аналитиков и разработчиков. Команда несет коллективную ответственность за процесс разработки.

При выполнении этих условий использование Scrum становится возможно. Профиль методологии Scrum, нанесенный на диаграмму параметров проекта представлен на Рисунок 3-4.

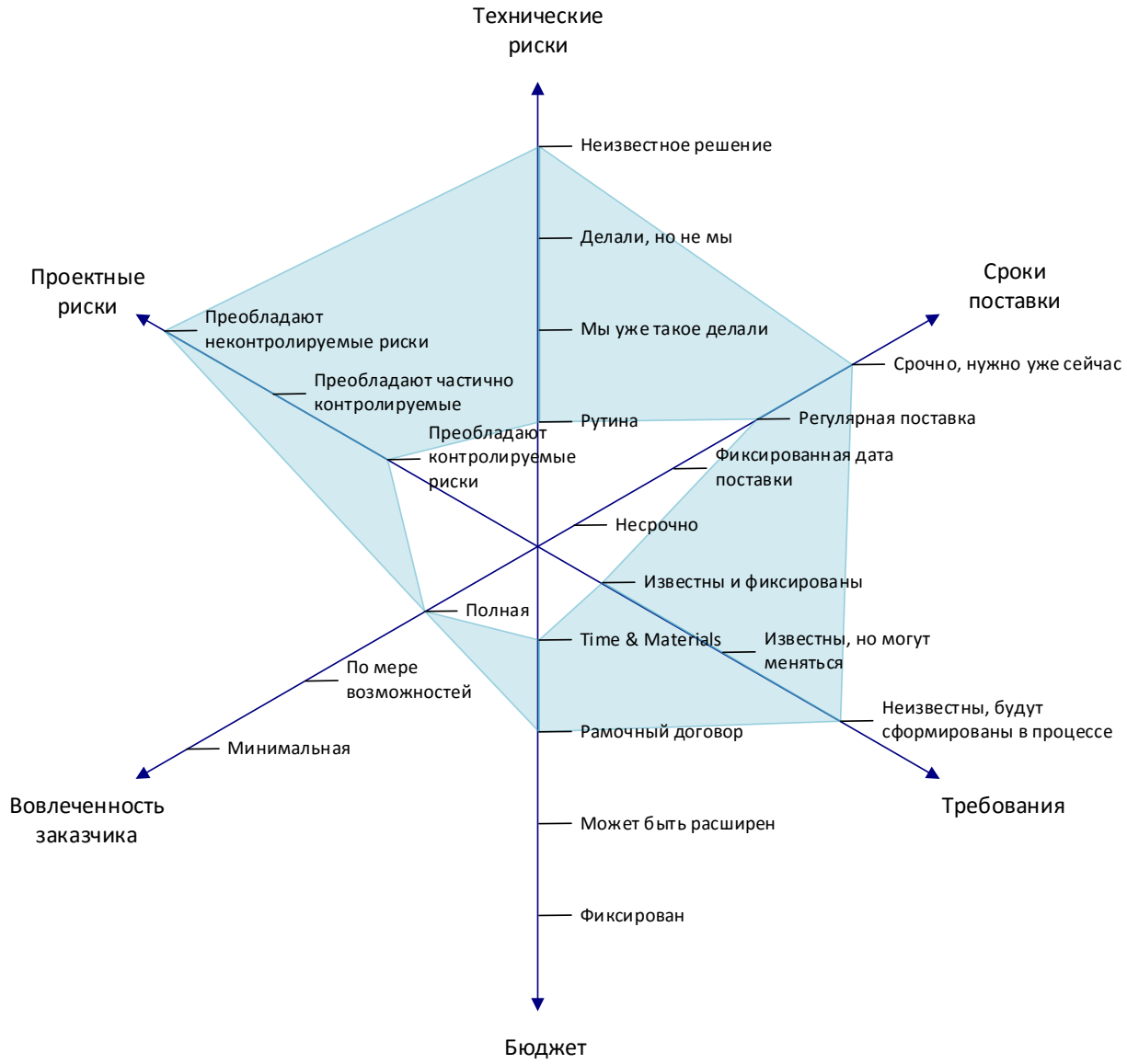

Рисунок 3-4. **Профиль методологии Scrum на диаграмме параметров проекта**

Источник: собственная разработка

Методология Kanban

Kanban используется для проектов для заказчиков, с которыми уже имеется обширный опыт работы и налажены проектные процессы с обоих сторон.

Заказчик доволен сотрудничеством с компанией и заинтересован в результативности, зачастую такой вид сотрудничества можно назвать аутсорсингом.

Имеется каталог приоритизированных задач- бэклог, составленных заказчиком. Команда разработки «вытягивает» из каталога задачи и концентрируется на текущей работе, при завершении рабочей задачи команда забирает следующую с верха бэклога. Владелец продукта может менять приоритизацию задач в бэклоге, поэтому в работу всегда идут наиболее важные задачи.

Количество задач, выполняемых командой одновременно ограничено. Это ограничение необходимо для управления скоростью производства, а также скоростью реагирования на изменения плана.

Профиль методологии Kanban, нанесенный на диаграмму параметров проекта представлен на Рисунок 3-5.

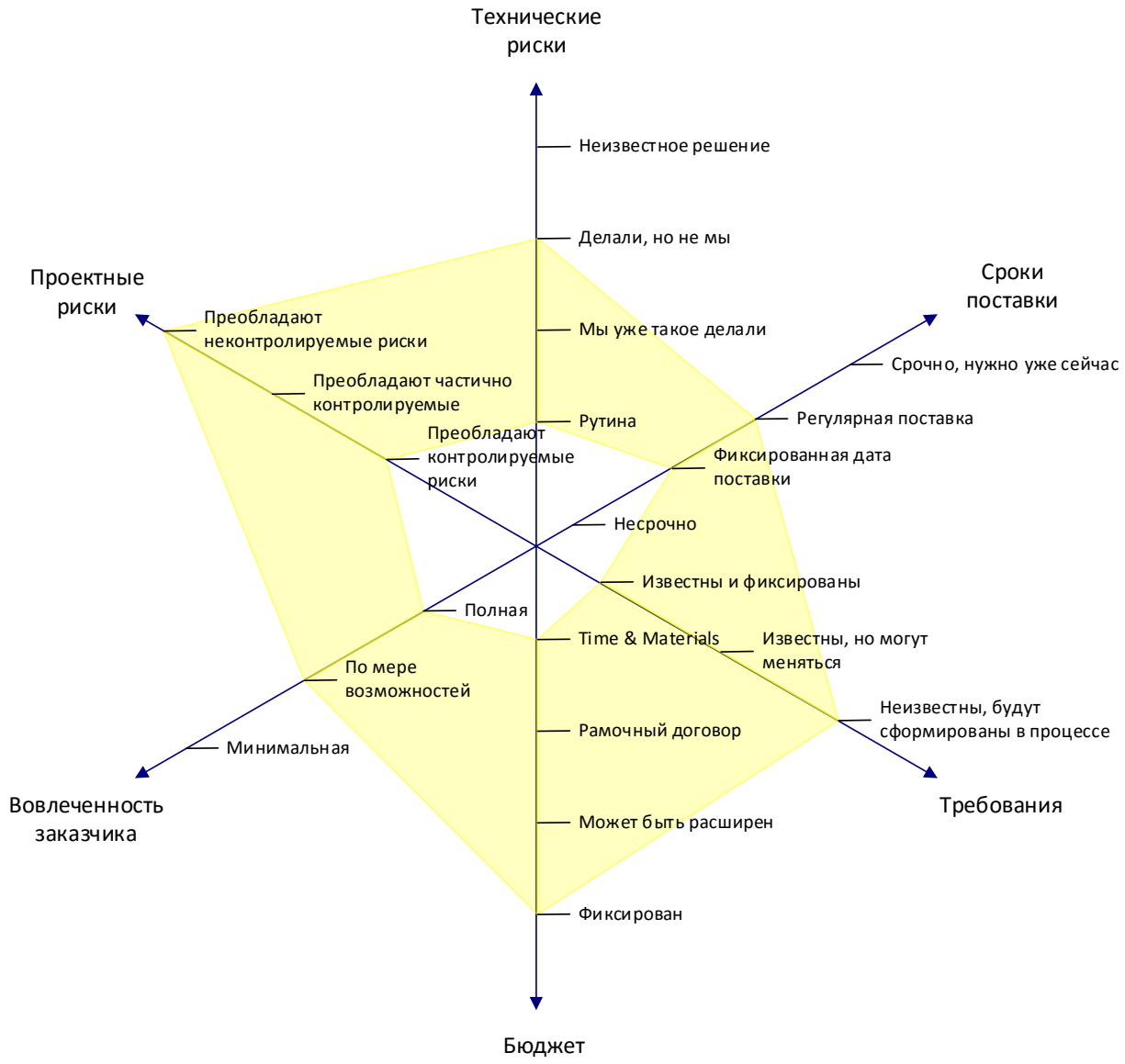

Рисунок 3-5. **Профиль методологии Kanban на диаграмме параметров проекта**

Источник: собственная разработка

Модель профилей методологий на диаграмме параметров проектов создана. Для наглядности отображения покрытия методологиями параметров проектов можно объединить все диаграммы в одну (Рисунок 3-6).

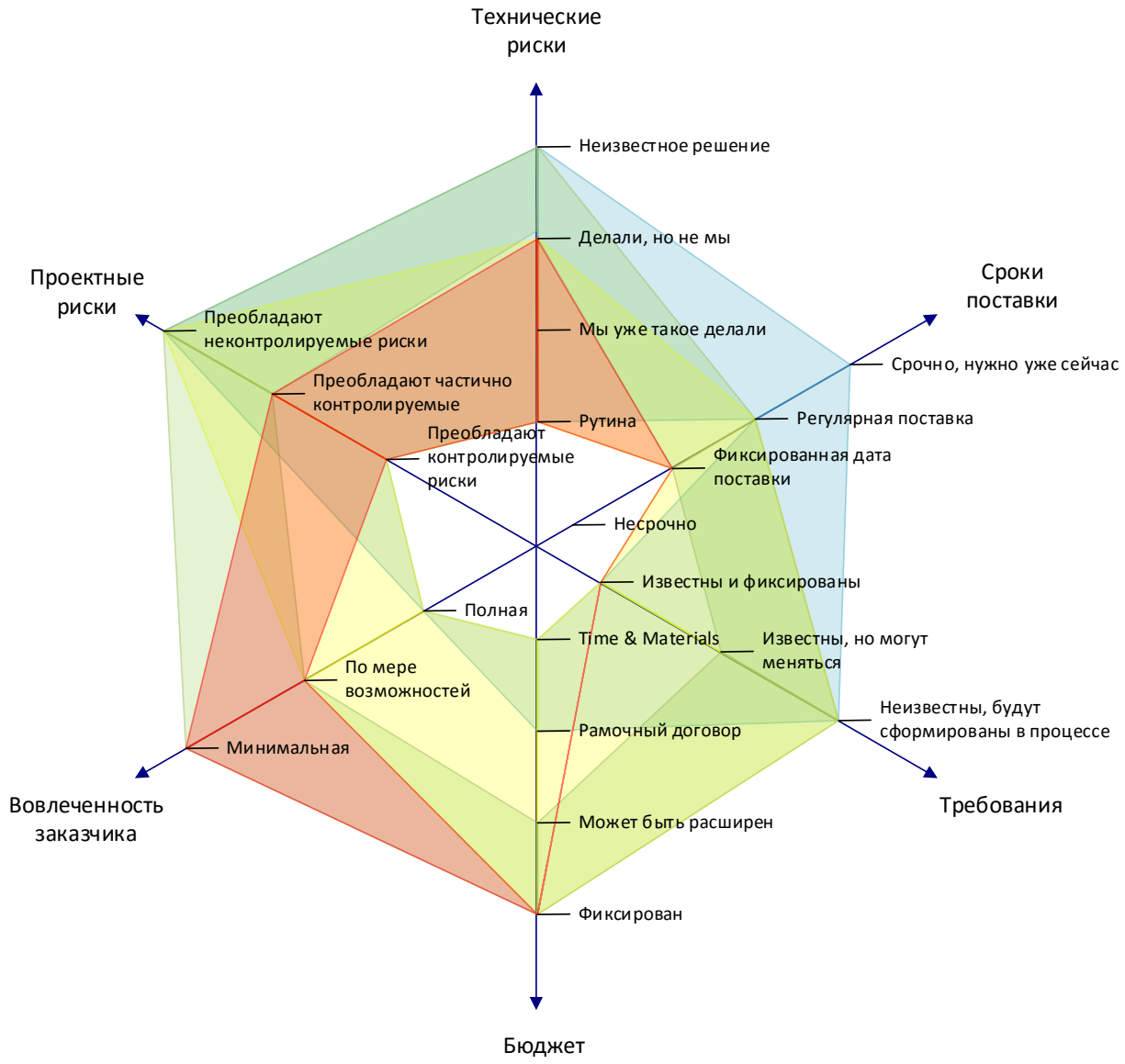

Рисунок 3-6. **Сводная диаграмма методологий компании Qlever Solutions**

Источник: собственная разработка

3.3 Создание модели профиля проекта на диаграмме параметров проектов

Следующий шаг: необходимо проанализировать имеющиеся проекты, соотнести с параметрами и их значениями и нанести профили проектов на диаграмму.

Для этого требуется собрать еще одну экспертную группу, в которую помимо уже имеющихся специалистов необходимо включить следующих:

- Менеджер проекта со стороны заказчика;
- Предполагаемые пользователи ПО;
- Сторонники продукта со стороны заказчика.

На основе имеющихся знаний о задаче у сотрудников со стороны заказчика, а также опыта специалистов команды разработки, создается профиль проекта в рамках имеющихся параметров и наносится на диаграмму параметров проекта.

В качестве примера для анализа на предмет выбора методологии разработки были выбраны два актуальных проекта компании Qlever Solutions:

- Проект «BI Аэро» - внедрение системы аналитической отчетности QlikView в АО «Газпром нефть-Аэро»;
- Проект «АРМ ДИТАТ» - разработка автоматизированного рабочего места Начальника ДИТАТ ПАО «Газпром нефть».

Эти проекты были выбраны по нескольким причинам:

1. на момент апробации модели оба проекта были на начальном этапе и требовалось определить подход к их реализации;
2. оба проекты значительно отличаются друг от друга;
3. подразумевается личное участие в обоих проектах, что позволяет быть вовлеченным в процесс тестирования состоятельности модели.

Проект «BI Аэро»

Проект направлен на повышение эффективности операционной деятельности бизнеса по задачам Планово-экономического управления и Управления маркетинга, а также на автоматизацию формирования отчетности, используемой различными направлениями компании, что достигается за счет выполнения следующих бизнес задач:

1. Снижение трудозатрат при формировании отчётности;
2. Повышение эффективности принятия управленческих решений;
3. Повышение качества и достоверности данных.

Выполнение указанных бизнес-задач позволит достичь следующих выгод:

1. Повышение достоверности и сходимости различных показателей отчетности

2. Снижение трудозатрат при формировании отчетности

3. Повышение эффективности принятия управленческих решений

4. Возможность перевода аналитики периметра ГПН-Аэро на системные источники данных.

5. Возможность использования встроенных в BI-платформу статистических методов и инструментов прогнозирования данных на следующем этапе развития.

Проект был получен на тендерной основе, это первое сотрудничество с заказчиком, поэтому обе стороны не знают, что ожидать друг от друга.

Таблица 5.

Параметры проекта "BI Аэро"

Параметр	Значение	Комментарий
Технические риски	Мы уже такое делали	Подобный проект уже был реализован в других дочерних компаниях
Сроки поставки	Фиксированная дата поставки	Из условия тендера
Требования	Известны и фиксированы	Из условия тендера
Бюджет	Фиксирован	Из условия тендера
Вовлеченность заказчика	По мере возможностей	Сотрудники со стороны заказчика проявляют интерес, но имеют высокую занятость по основным должностным обязанностям
Проектные риски	Преобладают контролируемые риски	Из условий тендера и оценки рисков менеджера проектов

Проект «АРМ ДИТАТ»

Цель проекта - разработка и внедрение информационно-аналитической системы «АРМ начальника ДИТАТ» для обеспечения качественной и оперативной информацией о показателях деятельности функции ИТАТ для своевременного принятия управленческих решений.

Задачи:

- Уточнение набора ключевых показателей, необходимых для своевременного принятия управленческих решений.
- Разработка графического дизайна Системы для визуализации показателей на мобильных устройствах.
- Разработка единой модели данных для хранения информации о показателях, разработка хранилища данных, инфо кубов и отчетов.

Заказчик уже имеет плодотворный опыт сотрудничества с компанией, поэтому был заключен рамочный договор по оказанию услуг разработки ПО.

Таблица 6.

Параметры проекта "АРМ ДИТАТ"

Параметр	Значение	Комментарий
Технические риски	Делали, но не мы	Подобный проект был реализован в других дочерних предприятиях внутренним сервисом компании
Сроки поставки	Срочно, нужно уже сейчас	Заказчик проявляет повышенный интерес к проекту, требуются частые релизы с обновлениями
Требования	Неизвестны, будут сформированы в процессе	Есть общее понимание цели и задач, но требования будут формироваться и расширяться в процессе проектной работы

Продолжение таблицы 5

Бюджет	Рамочный договор	Заказчик не впервые сотрудничает с компанией, удовлетворен результатом работ, поэтому был заключен рамочный договор на оказание услуг разработки
Вовлеченность заказчика	По мере возможностей	Сотрудники со стороны заказчика проявляют интерес, но имеют высокую занятость по основным должностным обязанностям
Проектные риски	Преобладают частично контролируемые	Существуют риски со стороны заказчика: неопределенность требований, доступность источников данных, приоритизация задач

Перенесем имеющиеся данные на диаграмму параметров проектов (Рисунок 3-7)

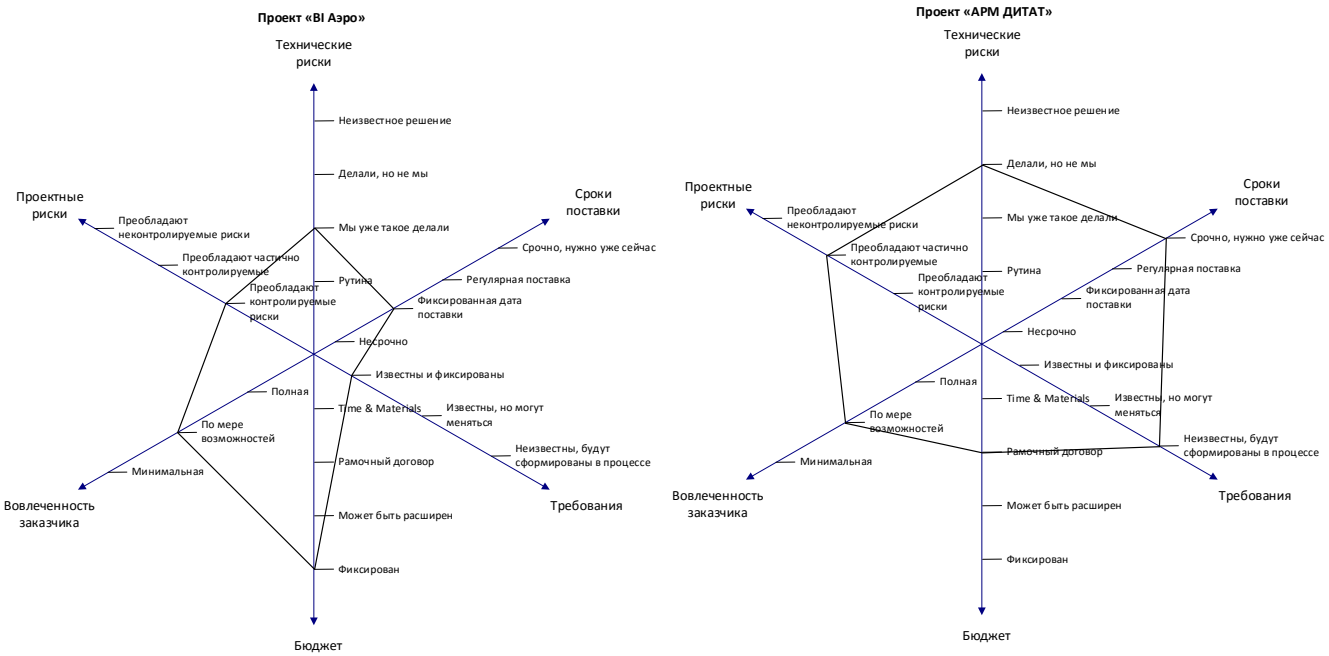

Рисунок 3-7. **Диаграмма профилей проектов «АРМ ДИТАТ» и «BI Аэро»**

Источник: собственная разработка

3.4Создание модели выбора методологии разработки программного обеспечения

Были получены две модели:

- Модель профилей методологий на диаграмме параметров проектов;
- Модель профилей проектов на диаграмме параметров проектов.

Следующий шаг – объединить эти модели и проанализировать попадание профиля проекта в профиль той или иной методологии. В случае, если профилей методологий немного, то появляется возможность использовать сводную модель методологий.

На Рисунок 3-8 представлена итоговая модель выбора методологии разработки для проекта «BI Аэро».

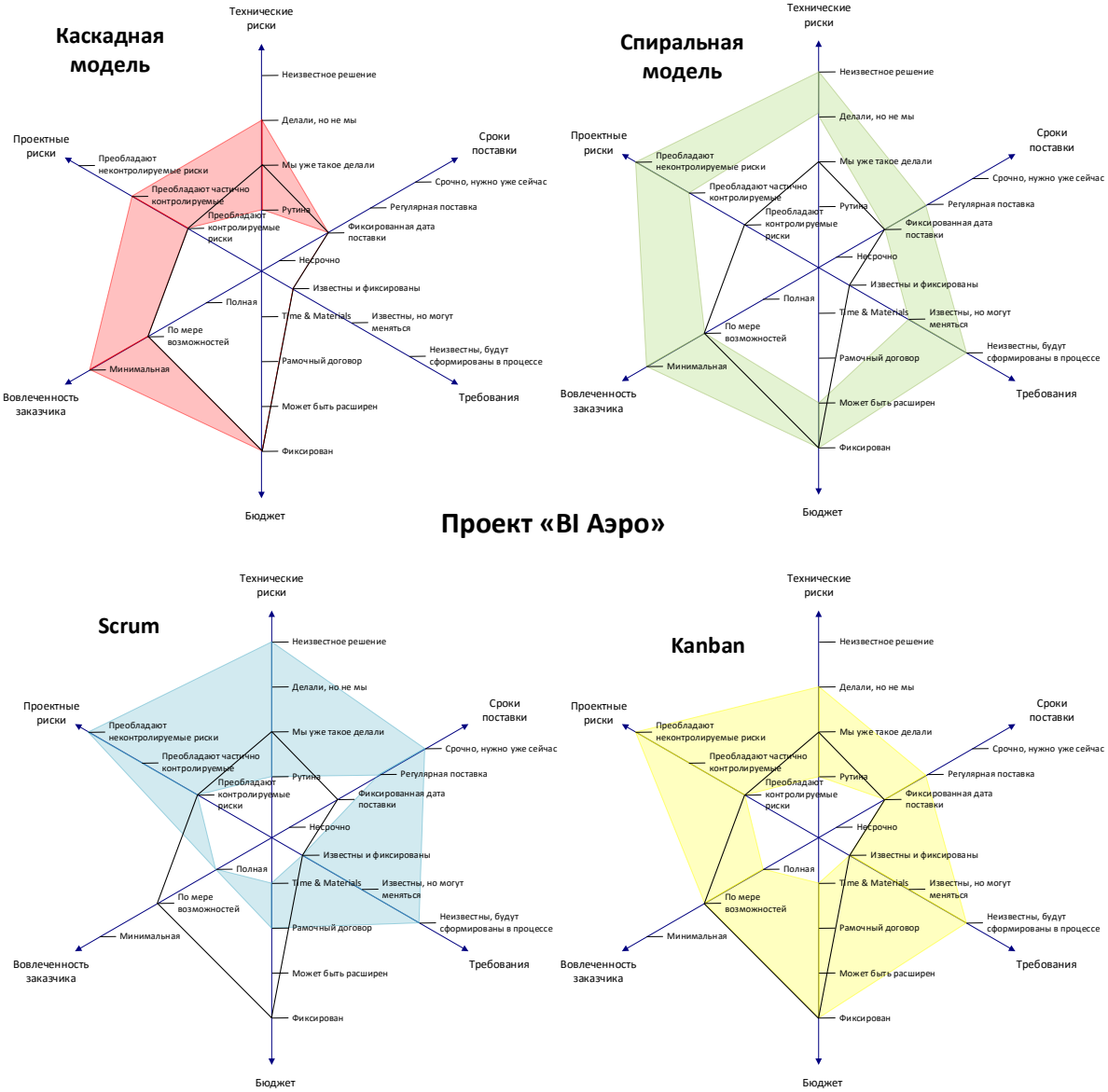

Рисунок 3-8. **Модель выбора методологии разработки для проекта "BI Аэро"**

Источник: собственная разработка

Модель позволяет наглядно показать соответствие параметров имеющегося проекта параметрам той или иной методологии.

Например, для «BI Аэро» хорошо подходят каскадная модель и методология Kanban. На основе имеющейся модели руководители проектов могут принять решение насчет выбора этих методологий и защитить решение перед высшим руководством.

На примере этого проекта была выбрана каскадная модель, так как заказчик ранее не работал с компанией и не знаком с процессом проектной работы по методологии Kanban. Кроме того, тендер фиксировал требования, сроки и бюджет, преобладали контролируемые проектные риски – имело смысл использовать более простую в использовании каскадную модель.

На Рисунок 3-9 представлена модель выбора методологии для проекта «АРМ ДИТАТ».

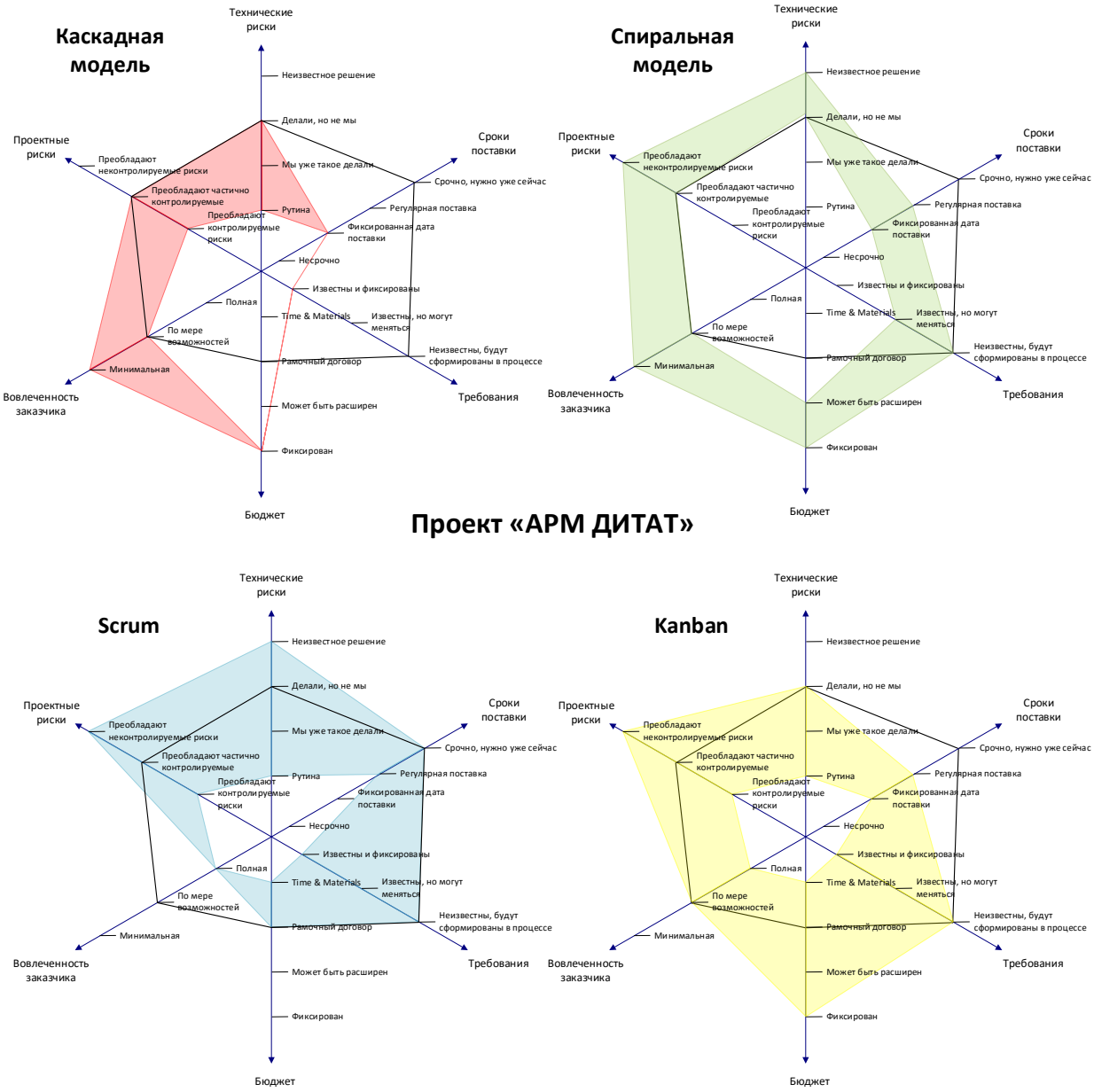

Рисунок 3-9. **Модель выбора методологии разработки для проекта "АРМ ДИТАТ"**

Источник: собственная разработка

По модели видно, что проект «АРМ ДИТАТ» имеет по одному высоко рисковому параметру при использовании методологий Scrum и Kanban.

В случае использования Scrum – появляется риск неполной занятости сотрудников со стороны заказчика, что может существенно отразиться на процессе проектной работы.

При использовании Kanban существует риск недопоставки обновленных версий заказчику, что не соответствует его интересам.

Проанализировав модель, руководитель проекта совместно с заказчиком приняли решение использовать Kanban, так как риск не предоставления очередного прототипа менее

критичен, по сравнению с риском замедления процесса разработки или недостаточной проработки требований.

При желании можно совместить все профили проектов и методологий на одной диаграмме, только необходимо учитывать их количество и удобство такого представления. Сводная модель выбора методологии разработки ПО для проектов «АРМ ДИТАТ» и «BI Аэро» представлена на Рисунок 3-10.

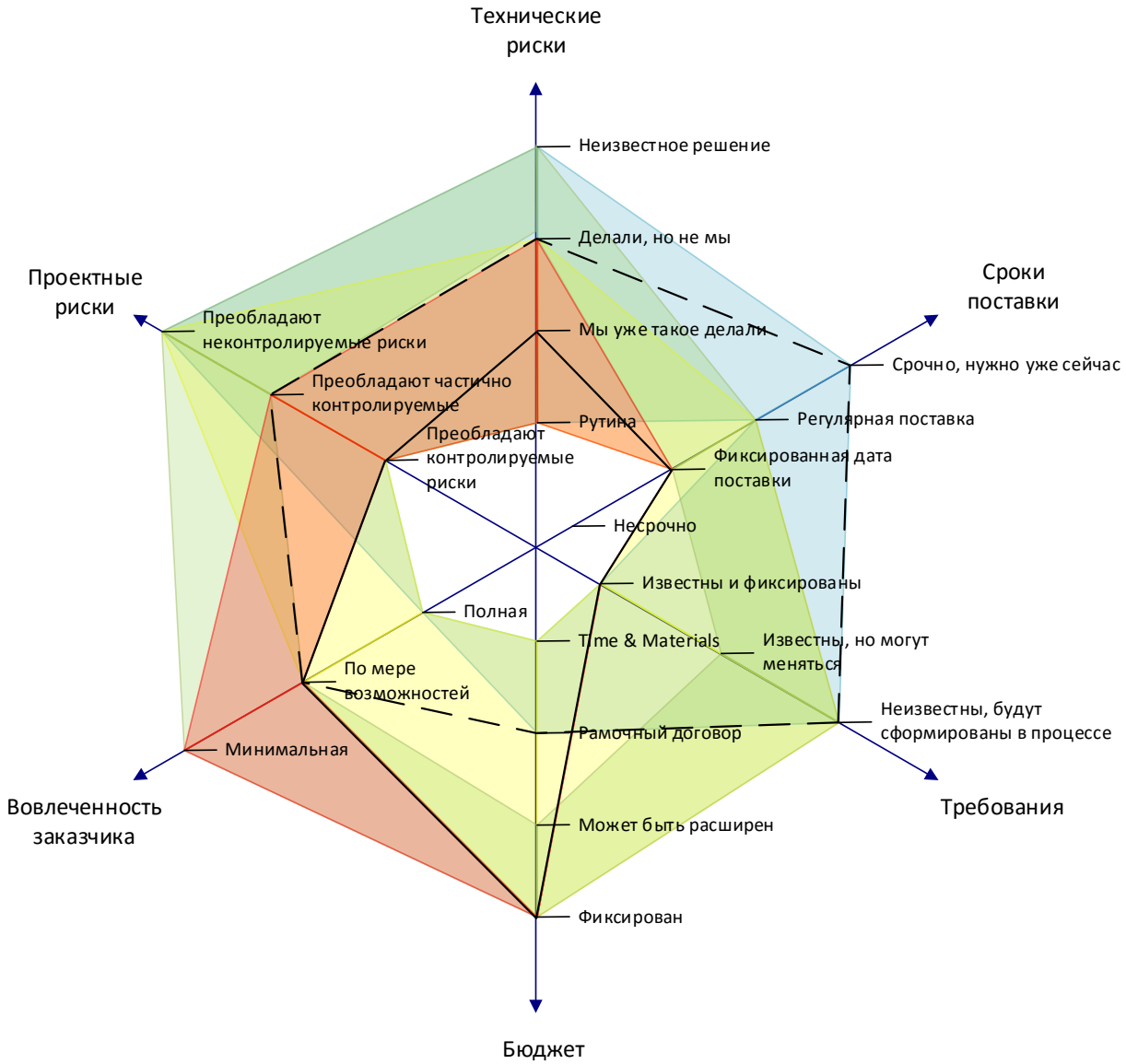

Рисунок 3-10. **Сводная модель выбора методологии разработки ПО для проектов «АРМ ДИТАТ» и «BI Аэро»**

Источник: собственная разработка

Пунктирной линией обозначен проект «АРМ ДИТАТ», непрерывной – «BI Аэро».

Таким образом была создана модель выбора методологий разработки программного обеспечения для компании Qlever Solutions.

Выводы по главе

В результате практического применения предложенной модели выбора в компании Qlever Solutions были получены следующие результаты:

1. **Использование предложенной модели позволяет существенно снизить риск выбора неподходящей для проекта методологии разработки.**

В свою очередь, выбор неподходящей методологии разработки влечет за собой как минимум риск срыва сроков проекта, как максимум риск закрытия проекта из-за возросших финансовых требований или невыполнения обязательств перед заказчиком.

К примеру, из-за того, что процесс взаимодействия с заказчиком ожидалось построить одним образом, но оказалось, что сотрудники со стороны заказчика оказались недостаточно вовлечены или загружены основной работой, были выявлены не все требования к программному обеспечению, разработчикам пришлось вносить существенные изменения в уже переведенный в опытно-промышленную эксплуатацию продукт. Цена исправления такой ошибки очень высока и в финансовом плане, и в плане ресурсоемкости, но этой ситуации можно было избежать, если бы для выбора методологии разработки использовался инструмент, позволяющий анализировать такие риски.

Кроме того, выбор неподходящей методологии может в значительной степени усложнить и замедлить процессы работы над проектом. Так, неинициативный руководитель может принять решение работать над проектом по хорошо ему известной и простой для использования каскадной модели. При непосредственном контакте команды с заказчиком выясняется, что заказчик – замотивирован и нацелен на результат, готов взаимодействовать с командой разработки ради достижения наилучшего результата. В такой ситуации возможности по совместной работе и способам взаимодействия у команды очень ограничены, процесс работы осложнен документацией. Выбранная методология не удовлетворяет ни заказчика, ни команду разработки, процесс разработки можно было бы организовать эффективнее и выгоднее для обоих сторон.

2. **Использование модели позволяет учитывать мнения заказчика и компетентных специалистов при принятии решения о выборе методологии разработки.**

Это позволяет распределить ответственность за принятие решения между всеми участниками, таким образом, во-первых, решение становится более объективным, так как будут учитываться мнения каждой стороны, во-вторых, ответственность за неверное решение будут нести коллективно все участники, а не только исполнитель.

Коллективная ответственность позволяет минимизировать риск принятия неверного решения, но и в случае, если риск реализуется, ответственность за него несут все лица, участвовавшие в создании модели, а не только руководитель проекта или исполнитель.

Таким образом, на примере использования в компании Qlever Solutions выявлено, что модель способствует:

- Повышению эффективности построения процесса разработки;
- Снижению рисков невыполнения сроков разработки;
- Повышению прозрачности процесса выбора методологии разработки, что ведет к росту лояльности заказчика;
- Повышению объективности и обоснованности решения о выборе той или иной методологии;
- Уменьшению риска выбора неподходящей для конкретного случая методологии, что может стать источником возникновения комплекса дополнительных рисков и проблем;
- Снижению риска появления незапланированных финансовых затрат;
- Снижению риска преждевременного закрытия проекта, связанного с неудовлетворенностью заказчика работой исполнителя.

Заключение

Предложенная в данной работе модель позволяет структурировать и формализовать процесс выбора методологии разработки программного обеспечения.

В ходе апробации модели было выявлено:

1. Состоятельность модели. Модель позволяет формализовать процесс выбора методологии для проекта, сделать его прозрачным и понятным каждому члену команды;
2. Модель позволяет провести объективный и обоснованный анализ сопоставимости методологий разработки и проектов разных типов, который будет включать мнения экспертов как со стороны заказчика, так со стороны исполнителя;
3. Модель позволяет значительно снизить риск принятия неверного решения выбора методологии разработки и связанные с этим риски. Теперь решения руководителей проектов компании могут основываться не только на интуиции и личном опыте, но и на формализованной модели, полученной при помощи совместного анализа проектной команды, что подразумевает коллективную ответственность за выбор той или иной методологии;
4. Модель зарекомендовала себя как инструмент для наглядной демонстрации и защиты выбора методологии разработки перед вышестоящим начальством и заказчиками. Пошаговый алгоритм построения модели позволяет детально рассматривать процесс формирования решения по использованию той или иной методологии, он прост в усвоении и понятен даже неподготовленному специалисту.

В работе проводится анализ специализированной и научной литературы, посвященной исследованиям методологий разработки программного обеспечения.

Определяется понятийный аппарат предметной области, после чего проводится подробный обзор и анализ некоторых популярных методологий разработки программного обеспечения, их преимуществ и критериев использования.

Исследуются предпосылки использования методологий разработки, рассматривается концепция Кеневин, позволяющая определить критерии оценки проектов по разработке программного обеспечения и классифицировать их. Предлагается и теоретически описывается модель, позволяющая анализировать сопоставимость той или иной методологии с различными проектами.

Приводится результат практического применения разработанной модели в качестве инструмента поддержки принятия решения руководителя проекта в компании Клевер

Солюшнс. Проводится анализ результатов применения модели и дана оценка ее состоятельности и применимости в реальной проектной деятельности.

Научная новизна исследования заключается в систематизации имеющихся знаний в области методологий разработки программного обеспечения и выявлении новаторской модели, позволяющей формализовать и структурировать процесс выбора методологий разработки.

Практическая значимость работы обуславливается возможностью применения модели как инструмента поддержки принятия решения с целью повышения качества и эффективности процесса принятия решения в компаниях, занимающихся разработкой программного обеспечения.

Перспективы дальнейшего развития модели заключаются в возможности создания автоматизированного инструмента визуализации модели, а также в исследовании количественной или стоимостной оценки выбора той или иной методологии.

Список литературы

Статьи

1. Аникеев Д.А., Пешкова К.Е., Гарченко Е.В., Сарапулова Т.В. Обзор методологий разработки программного обеспечения // СОВРЕМЕННЫЕ ТЕНДЕНЦИИ РАЗВИТИЯ НАУКИ И ПРОИЗВОДСТВА. – 2016. – С. 264-266.
2. Борцов М. Ю., Молочников М. А. Гибкий подход к разработке и внедрению информационных систем предприятия // Известия Санкт-Петербургского государственного электротехнического университета ЛЭТИ. – 2005. – №. 11. – С. 42-48.
3. Бурбело С.М., Стародуб О.С., Богданова М.С. Выбор Гибких Методов Разработки Программного Обеспечения // Вісник Хмельницького Національного Університету. Технічні Науки. 2013. № 4 (203). С. 139–143.
4. Восканян Л.С., Искандарян Д.С. Применение методологии Scrum в передовых компаниях // BI-ТЕХНОЛОГИИ В ОПТИМИЗАЦИИ БИЗНЕС-ПРОЦЕССОВ. Материалы Международной научно-практической очно-заочной конференции. Российско-Армянский (Славянский) университет, Уральский государственный экономический университет. - 2014. С. 31–35.
5. Говорущенко Т.О., Малярчук Р.А. Анализ Процесса Выбора Технологии Проектирования, Методологии И Среды Разрабатывания Программного Обеспечения // Вісник Хмельницького Національного Університету. Технічні Науки. 2014. № 6 (219). С. 186–195.ДЖЕЛДУБАЕВ Р.С. Применение современных методологий разработки программного обеспечения в учебном проекте / Р.С. ДЖЕЛДУБАЕВ, Ю.П. МОСКАЛЕВА // ПРОБЛЕМЫ СОВРЕМЕННОГО ПЕДАГОГИЧЕСКОГО ОБРАЗОВАНИЯ. – 2015. – № 48-4. – С. 33-44.
6. Евсеев Л. В. ПРОБЛЕМЫ АДАПТАЦИИ SCRUM-ИНСТРУМЕНТОВ В РОССИЙСКОЙ ПРАКТИКЕ УПРАВЛЕНИЯ ИТ-ПРОЕКТАМИ //Экономическое развитие России: тенденции, перспективы. – 2015. – С. 213-215.
7. Ильясова Ф. С., Клеблеев Ш. А. Современные методологии разработки программного обеспечения //Инновационные направления в научной и образовательной деятельности. – 2015. – С. 76-78.
8. Кауров А.А., Кривцов А.Н. Имитационное моделирование при оценке проектной деятельности предприятия // ПЕРСПЕКТИВЫ РАЗВИТИЯ НАУКИ В СОВРЕМЕННОМ МИРЕ. Сборник статей по материалам V международной научно-практической конференции : В 3 частях. - 2018. С. 71–76.

9. Карпов Д. В. Гибкая методология разработки программного обеспечения //Вестник Нижегородского университета им. НИ Лобачевского. – 2011. – №. 3-2.
10. Лим С.А. Исследование эффективности работы скрама при работе с малыми проектами // Новые Информационные Технологии В Автоматизированных Системах. 2015. № 18. С. 476–478.
11. Матвиенко Д.Н. Инновационные методологии перепроектирования «унаследованных» программных систем // Омский Научный Вестник. 2009. № 3 (83). С. 249–251.
12. Мерзлякова А.П. Высокотехнологичное предприятие как субъект инновационно-креативной деятельности // В сборнике «Проблемы развития инновационно-креативной экономики». — 2011 — с. 225-233.
13. Михайлов А. А., Базуева С. А. АНАЛИЗ СТРУКТУРНО-ПАРАМЕТРИЧЕСКОЙ ИНТЕГРАЦИИ АЛГОРИТМОВ //ТЕХНОЛОГИИ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ ТРИС-2016. – 2016. – С. 128-133.
14. Овчинников С. А. Управление проектом по разработке программного обеспечения с целью повышения качества на основе анализа проектных рисков //Известия Волгоградского государственного технического университета. – 2013. – Т. 16. – №. 8 (111).
15. Павленко Е. П., Криворотенко И. А., Айвазов В. А. Выбор методологии разработки программного обеспечения для страховой компании. – 2014.
16. Петрова А. Н., Еськова А. В., Лошманов А. Ю. Проблема выбора методологии разработки информационной системы вуза //Современные проблемы науки и образования. – 2013. – №. 2. – С. 534-534.
17. Плотников А. Н. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОИЗВОДСТВА И ПЕРСПЕКТИВНЫЕ МОДЕЛИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ //Гетеромагнитная микроэлектроника. – 2012. – №. 13. – С. 123-130.
18. Сачек Е. А. Применение имитационного моделирования в управлении проектами по разработке программного обеспечения с гибкими методологиями //Научные труды Северо-Западного института управления. – 2015. – Т. 6. – №. 4. – С. 251-257.
19. Сербская О. В. Применение гибких методологий управления проектами в образовательной деятельности //Социальная политика и социология. – 2015. – Т. 14. – №. 6. – С. 132.
20. Свиридова А. С. и др. Анализ современных методолгогий разработки сложных программных проектов //Актуальные проблемы авиации и космонавтики. – 2015. – Т. 1. – №. 11.

21. Геркушенко Г. Г., Ткаченко А. В. Сравнительный анализ методологий разработки программного обеспечения //Наука, техника и образование. – 2016. – №. 3. – С. 109-113.
22. Тюнина А.И., Беляев Р.В. О гибких методологиях при разработке программного обеспечения // ИНФОРМАТИКА: ПРОБЛЕМЫ, МЕТОДОЛОГИЯ, ТЕХНОЛОГИИ. Материалы XVI Международной научно-методической конференции. - 2016. С. 178–182.
23. Abrahamsson P. et al. Agile software development methods: Review and analysis //arXiv preprint arXiv:1709.08439. – 2017.
24. Ahmad M. O. et al. Kanban in software engineering: A systematic mapping study //Journal of Systems and Software. – 2018. – Т. 137. – P. 96-113.
25. Boehm B., A spiral model of software development and enhancement, ACM SIGSOFT Software Engineering Notes, v.11 n.4, p.14-24, August 1986
26. Dingsøyr T., Moe N. B. Towards principles of large-scale agile development //International Conference on Agile Software Development. – Springer, Cham, 2014. – P. 1-8.
27. French S. Cynefin: uncertainty, small worlds and scenarios //Journal of the Operational Research Society. – 2015. – Т. 66. – №. 10. – P. 1635-1645.
28. Kumar G., Bhatia P. K. Comparative analysis of software engineering models from traditional to modern methodologies //Advanced Computing & Communication Technologies (ACCT), 2014 Fourth International Conference on. – IEEE, 2014. – P. 189-196.
29. Steinhardt G. Agile Software Development //The Product Manager's Toolkit®. – Springer, Cham, 2017. – P. 131-147.
30. Tripathi N. et al. Scaling Kanban for software development in a multisite organization: challenges and potential solutions //International Conference on Agile Software Development. – Springer, Cham, 2015. – P. 178-190.
31. Turk D., France R., Rumpe B. Assumptions underlying agile software development processes //arXiv preprint arXiv:1409.6610. – 2014.
32. Turk D., France R., Rumpe B. Limitations of agile software processes //arXiv preprint arXiv:1409.6600. – 2014.
33. Vijayasarathy L. R., Butler C. W. Choice of software development methodologies: Do organizational, project, and team characteristics matter? //IEEE Software. – 2016. – Т. 33. – №. 5. – P. 86-94.

Книги

34. Ауэр К., Миллер Р. Экстремальное программирование. Постановка процессов: пер. с англ. СПб., 2004.
35. Липаев, В.В. Программная инженерия: методологические основы: учебник / В.В. Липаев. – М.-Берлин: Директ Медиа, 2015. - 608 с.
36. Расмуссон Д. Гибкое управление IT-проектами. Руководство для настоящих самураев. СПб., 2012.
37. Cockburn A. Agile software development. – Boston : Addison-Wesley, 2002. – Т. 177.
38. Leopold K. Practical Kanban: From Team Focus to Creating Value / K. Leopold, LEANability PRESS, 2017. 354 p.
39. Sutherland J., Sutherland J.J. Scrum: The Art of Doing Twice the Work in Half the Time / J. Sutherland, J.J. Sutherland, 1st edition., New York: Currency, 2014. 256 p.
40. Schwaber K., Beedle M. Agile software development with Scrum. – Upper Saddle River : Prentice Hall, 2002. – Т. 1.
41. Sommerville I. Software Engineering: United States Edition / I. Sommerville, 9 edition., Boston: Pearson, 2010. 792 p.
42. Stephens R. Beginning Software Engineering / R. Stephens, 1 edition., Sybex, 2015. 480 p.
43. Hans van Vliet, Software Engineering: Principles and Practice, 3rd edition, John Wiley & Sons, 2008

Интернет-ресурсы и электронные базы данных

44. 11-й ежегодный отчет State of Agile [Электронный ресурс]. URL: https://scrumtrek.ru/blog/11-j-ezhegodnyj-otchet-state-of-agile (дата обращения: 23.03.2018)
45. Agile Changer: History of Waterfall model [Электронный ресурс]. URL: http://www.agilechanger.com/2013/08/history-of-waterfall-model (дата обращения: 02.11.2017)
46. Agilemanifesto.org – портал создателей принципов Agile [Электронный ресурс]. URL: http://agilemanifesto.org/principles.html/ (дата обращения: 02.11.2017)
47. Craig Larman’s official web site [Электронный ресурс]. URL: http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf (дата обращения: 01.11.2017)
48. IEEE Computer Society [Электронный ресурс]. URL: https://www.computer.org/web/swebok/v3 (дата обращения: 01.11.2017)